

# Document Preparation with L<sup>A</sup>T<sub>E</sub>X

George Ferguson

The University of Rochester  
Computer Science Department  
Rochester, NY 14627

Departmental Guide 12

April 2007

## **Abstract**

This document describes how to use the Department's installation of L<sup>A</sup>T<sub>E</sub>X and related tools to produce documents. It is not a primer on how to use L<sup>A</sup>T<sub>E</sub>X. For that, see the L<sup>A</sup>T<sub>E</sub>X book referenced in the "Further Reading" section of this document. An appendix contains the "Local Guide" referred to in the L<sup>A</sup>T<sub>E</sub>X book, based on Leslie Lamport's original 1988 local guide updated and customized for our site.

---

The original version of Department Guide 12 was written by Lawrence Crowl and Ken Yap, based on Lamport's "local guide." Updated by George Ferguson in 1994 and 1998 and by Dan Gildea in 2006. Next version by you: `/usr/grads/doc/latex-guide/`.

# Contents

<b>1</b>	<b>Overview</b>	<b>3</b>
<b>2</b>	<b>Basic L<sup>A</sup>T<sub>E</sub>X</b>	<b>5</b>
2.1	Running L <sup>A</sup> T <sub>E</sub> X . . . . .	5
2.2	Using pdfL <sup>A</sup> T <sub>E</sub> X . . . . .	6
2.3	Where (and when) does L <sup>A</sup> T <sub>E</sub> X look for files? . . . . .	6
2.4	Re-running L <sup>A</sup> T <sub>E</sub> X: the .aux file . . . . .	7
2.5	Tables of contents and lists of figures and tables . . . . .	8
<b>3</b>	<b>Viewing the output</b>	<b>9</b>
3.1	Previewing DVI files under the X Window System . . . . .	9
3.2	Previewing problems . . . . .	9
<b>4</b>	<b>Postscript</b>	<b>11</b>
4.1	Postscript files and previewing Postscript . . . . .	11
4.2	Printing DVI . . . . .	11
4.3	Using Postscript fonts . . . . .	13
<b>5</b>	<b>Creating PDF</b>	<b>14</b>
<b>6</b>	<b>Creating and Including Figures</b>	<b>15</b>
6.1	L <sup>A</sup> T <sub>E</sub> X picture mode . . . . .	15
6.2	Postscript and <code>includegraphics</code> . . . . .	15
6.3	Tools for creating figures . . . . .	17
6.3.1	Xfig: Object-oriented diagrams . . . . .	17
6.3.2	Dia: Free-hand drawing . . . . .	18
6.3.3	GIMP: GNU Image Manipulation Program . . . . .	18
6.3.4	ImageMagick: General image processing . . . . .	19
6.3.5	Gnuplot: General purpose plotting . . . . .	19
6.3.6	Maxima: Interactive mathematical computation . . . . .	19
6.3.7	R: Statistical Computing . . . . .	20
6.3.8	Matlab: Matrix laboratory . . . . .	20
6.3.9	OpenOffice suite . . . . .	20
<b>7</b>	<b>Checking your spelling</b>	<b>21</b>
<b>8</b>	<b>Using BibT<sub>E</sub>X for bibliographic citations</b>	<b>23</b>

<b>9</b>	<b>Preparing slides for presentations</b>	<b>25</b>
<b>10</b>	<b>Editing L<sup>A</sup>T<sub>E</sub>X documents with AUC-T<sub>E</sub>X</b>	<b>27</b>
10.1	Accessing AUC-T <sub>E</sub> X . . . . .	27
10.1.1	Font Changes . . . . .	27
10.1.2	Environments and Sections . . . . .	28
10.1.3	Formatting, Previewing, Printing . . . . .	28
10.2	Other features . . . . .	29
<b>11</b>	<b>Further Reading</b>	<b>31</b>
<b>A</b>	<b>The L<sup>A</sup>T<sub>E</sub>X Local Guide</b>	<b>33</b>
A.1	Layout Parameters . . . . .	33
A.2	Document Styles . . . . .	36
A.2.1	The <code>proc</code> Style Option . . . . .	36
A.2.2	Double Spacing . . . . .	36
A.2.3	The <code>bezier</code> Style Option . . . . .	37
A.2.4	The <code>ifthen</code> Style Option . . . . .	37
A.2.5	Letters . . . . .	38
A.2.6	The <code>showidx</code> Style Option . . . . .	38
A.3	Where the Files Are . . . . .	38
A.4	Makeindex: An index processor for L <sup>A</sup> T <sub>E</sub> X . . . . .	38
A.4.1	How to Use <i>MakeIndex</i> . . . . .	38
A.4.2	How to Generate Index Entries . . . . .	39
A.4.3	Error Messages . . . . .	42
A.5	Fonts . . . . .	44
A.6	Summary of Available Programs . . . . .	45
A.7	Summary of L <sup>A</sup> T <sub>E</sub> XMacros . . . . .	46
A.8	Common Errors . . . . .	46
A.8.1	Header Commands . . . . .	47
A.8.2	Sections . . . . .	47
A.8.3	Spacing . . . . .	48
A.8.4	Figures and Tables . . . . .	48
A.9	Bugs . . . . .	49

# 1 Overview

$\text{T}_{\text{E}}\text{X}$  is a program that takes a text file describing a document and converts it into a device-independent page description language that can then be previewed or printed. This process is referred to as “formatting the document,” and  $\text{T}_{\text{E}}\text{X}$  is referred to as a “formatter.” Since it was developed by a computer scientist (Donald Knuth of Stanford), of course  $\text{T}_{\text{E}}\text{X}$  can be “programmed” (via macros) to add new features.  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  is a collection of macros for  $\text{T}_{\text{E}}\text{X}$  that are supposed to make certain common formatting operations easier. It has become the standard for technical writing in Computer Science. As of this writing, the “2e” version of  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  (written “ $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X} 2_{\epsilon}$ ”) has become the standard, replacing the older “2.09” version. This document uses simply  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  meaning  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X} 2_{\epsilon}$ , unless emphasizing a compatibility issue.

This document describes how to use  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  and its associated tools effectively. Figure 1 illustrates the process schematically, and it will be described in detail in what follows. Please note that this document does not describe how to create  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  documents. That is, it doesn’t describe the  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  language and its myriad special features, except insofar as they relate to issues involved in getting a document printed. If you aren’t familiar with  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ , you should start with the  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  book listed in the references (Section 11) or some other textbook on  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ . Whenever it refers you to your “local guide,” this document may come in useful.

University of Rochester Department of Computer Science readers will want to read Lawrence Crowl’s “Guide to Department Styles” (Department Guide 13) for details on available styles and macros. In particular, the styles required for department technical reports, dissertations, and thesis proposals are described there.

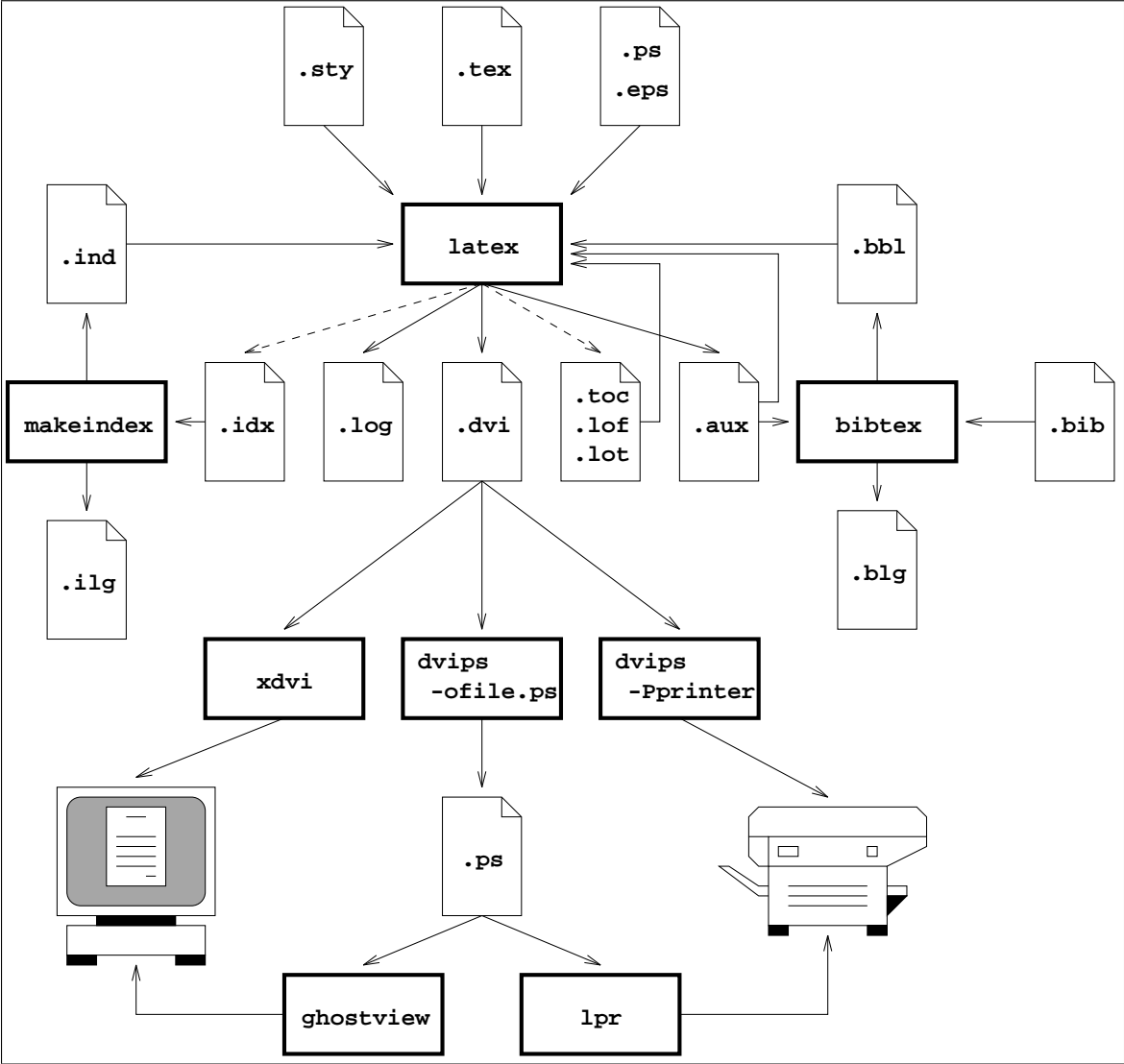


Figure 1: The  $\text{\LaTeX}$  Formatting Process

## 2 Basic L<sup>A</sup>T<sub>E</sub>X

This section describes enough about L<sup>A</sup>T<sub>E</sub>X to format simple documents.

### 2.1 Running L<sup>A</sup>T<sub>E</sub>X

Let's assume that you've read the L<sup>A</sup>T<sub>E</sub>X book and have the following input file named `paper.tex` ready to format.

```
\documentclass{article}
\begin{document}
Hello world.
\end{document}
```

To format the document, you give the command<sup>1</sup>

```
% latex paper
```

L<sup>A</sup>T<sub>E</sub>X will assume the `.tex` filename extension automatically. You should see output something like the following:

```
This is TeX, Version 3.14159 (C version 6.1)
(paper.tex
LaTeXe <1996/12/01>
Babel <v3.6h> and hyphenation patterns for american, loaded.
(/usr/grads/lib/teTeX/texmf/tex/latex/base/article.cls
Document Class: article 1996/10/31 v1.3u Standard LaTeX document class
(/usr/grads/lib/teTeX/texmf/tex/latex/base/size10.clo))
No file paper.aux.
[1] (paper.aux) )
Output written on paper.dvi (1 page, 232 bytes).
Transcript written on paper.log.
```

Several files will be generated automatically with the same basename as the input file (`paper` in this example) but with different extensions:

- `.dvi` This is the device-independent page description of your document, suitable for previewing or printing. It is a binary file and not meant for human consumption.
- `.log` This file contains a more detailed trace of L<sup>A</sup>T<sub>E</sub>X's execution than what was printed on the screen. You may want to check here if there were errors during formatting.
- `.aux` This is L<sup>A</sup>T<sub>E</sub>X's "auxiliary" output file, containing information determined during formatting such as page numbers, bibliographic tags, and more. It is discussed further below. You should never have to edit it.

---

<sup>1</sup>The `%` character in examples indicates your shell prompt (which may well be something else). You don't type the `%` symbol when entering shell commands.

The other files shown as outputs of  $\LaTeX$  in Figure 1 are only generated under certain conditions, and will be described when we come to those features.

In addition to the other stuff to be discussed later, you’ll see a list of numbers in square brackets like the [1] in the preceding example.  $\LaTeX$  prints the number after “cutting the scroll” and outputting that page to the .dvi file, so any messages you see apply to the *next* page to be output.

If you want to get on with things, you can skip to the next section and preview the document, rather than reading the rest of this section that describes something about how  $\LaTeX$  works and how it can affect you.

## 2.2 Using pdf $\LaTeX$

The pdf $\LaTeX$  program generates Adobe PDF output, rather than DVI format. Usually, PDF is what you ultimately want, and pdf $\LaTeX$  provides an easier solution than producing DVI, then converting to Postscript, and then to PDF. However, some  $\LaTeX$  packages make use of Postscript language features, requiring going through the chain described above. It can speed things up a lot to automate the process with a Makefile, see for example `/usr/grads/doc/latex-guide/Makefile`.

## 2.3 Where (and when) does $\LaTeX$ look for files?

One important thing to understand about  $\LaTeX$ ’s output is where the input is coming from. This information is contained in  $\LaTeX$ ’s output (and in the .log file).

Looking at the example trace,  $\LaTeX$  started by processing the input file `paper.tex`, which it found in the current directory. In general,  $\LaTeX$  prints an open parenthesis and the filename when it starts to process a file, and anything printed until the matching closing parenthesis occurred during the processing of that file.

Then, while processing `paper.tex`,  $\LaTeX$  included `article.sty`, which it found in one of the system directories. When you specify a document class using the `\documentclass` command,  $\LaTeX$  looks for a corresponding .cls file. Since we specified “article” style,  $\LaTeX$  included `article.cls`. Similarly, while processing `article.cls`,  $\LaTeX$  included `size10.clo`, which sets the default 10 point typesize. If you had specified a class option like `11pt` or `12pt`,  $\LaTeX$  would have included `size11.clo` or `size12.clo` instead. These are details of the “article” style that you don’t really need to be aware of, except to realize that the document class and any class options cause  $\LaTeX$  to search for and include additional input files.

By default  $\LaTeX$  looks for files in the current directory (“.” in Unix terminology) and in the system directory. You can change  $\LaTeX$ ’s search path by setting the environment variable `TEXINPUTS` to a colon-separated list of directories to search. A leading colon means to search the default directories first; a trailing colon means to search the default directories last. For example, you could add a directory containing personal macros by doing

```
% setenv TEXINPUTS /u/myname/tex:
```

With this setting,  $\LaTeX$  will search `/u/myname/tex`, then the current directory, then the system directory (since those last two are the default search path).

You will almost certainly want to end `TEXINPUTS` with a colon or  $\LaTeX$  will probably not find various basic files (like `article.cls`). You will almost certainly not want to list the system directories explicitly in `TEXINPUTS` since its location may change in the future. More details are in the manpage for `tex(1)`.

## 2.4 Re-running $\LaTeX$ : the `.aux` file

If you look at Figure 1 carefully, you'll notice that the `.aux` file produced by running  $\LaTeX$  is also an input to  $\LaTeX$ . In the example above, we got the message “No file `paper.aux`.” If you run  $\LaTeX$  again, you'll see that `paper.aux` is included immediately after the article style files are included. After the document is finished, `paper.aux` is written out again (which is indicated by the same parenthetical convention as for input, unfortunately).

The reason for this circularity is that  $\LaTeX$  uses the `.aux` file to communicate between multiple runs. That is,  $\LaTeX$  processes the document in one pass, outputting pages one by one. There may be things it doesn't know until later, such as what page a figure will appear on, or what page a section starts at. For example, change the sample file to add the following text after “Hello World.”

```
See Section~\ref{sect}.
\section{The Section}\label{sect}
That is all.
```

Now, when  $\LaTeX$  encounters the `\ref` command, it doesn't know what the section number is yet. You get the following message:

```
LaTeX Warning: Reference 'sect' on page 1 undefined on input line 4.
```

and  $\LaTeX$  substitutes “??” for the section number. Only when it gets to the `\label` command does it know what the section number is. In general, the page containing the reference might already have been output— $\LaTeX$  has no way to go back and “patch” the section number in.

Instead, what  $\LaTeX$  does is output things like labels to the `.aux` file. In the previous example, the `.aux` file contains, among other things, the line:

```
\newlabel{sect}{{1}{1}}
```

Then when you run  $\LaTeX$  a second time, the `.aux` file gets included at the outset, and this time when the reference is processed, the value of the label is known.

The moral of the story is that you often have to run  $\LaTeX$  more than once to get all the interdependencies ironed out. Sometimes twice isn't enough, as when replacing labels causes other things to move on the page, thereby changing their labels.  $\LaTeX$  tries to help you out by indicating in its output when it thinks labels may have changed.

Another moral is that sometimes  $\LaTeX$  will put something in the `.aux` file that only causes an error when that file is read in the next time. Common culprits are errors in captions, section headings, or table of contents entries. You can tell when this is the case by looking carefully in the `.log` file for which file was being processed when the errors occurred. You can always delete the `.aux` file and start again from scratch.

## 2.5 Tables of contents and lists of figures and tables

Some documents, notably dissertations, require a table of contents and lists of figures and tables. With  $\LaTeX$ , you can create these automatically by using the `\tableofcontents`, `\listoffigures`, and `\listoftables` commands as described in the  $\LaTeX$  book.

You place the command at the point in the document where you want the contents or list to appear. As described in the  $\LaTeX$  book and similar to the use of the `.aux` file described above, this causes  $\LaTeX$  to do two things. First, it causes  $\LaTeX$  to write a file whose name ends in `.toc` for the table of contents, `.lof` for the list of figures, or `.lot` for the list of tables.  $\LaTeX$  will automatically add an entry to the appropriate file for each `\chapter`, `\section`, *etc.*, command for the table of contents, and for each `figure` or `table` environment for the lists, respectively. Second,  $\LaTeX$  reads the appropriate file from the previous run. So if you run  $\LaTeX$  twice, the first time will generate the file and the second time will include it in your document (and, incidentally, re-write the file). As noted above, including the new files may cause other things to change, such as page numbers if the table of contents takes up several pages, in which case you may have to run  $\LaTeX$  again to get everything lined up.

## 3 Viewing the output

At this point you should have successfully run  $\text{\LaTeX}$  and you should have a `.dvi` file (`paper.dvi` in our example) containing its output.

### 3.1 Previewing DVI files under the X Window System

For previewing `.dvi` files on workstations you use the program `xdvi`. As with  $\text{\LaTeX}$ , you run it and pass the basename of the file you want to preview on the command line:

```
% xdvi paper
```

A variety of other options are detailed in the manpage `xdvi(1)`. Most of them are unnecessary, although `-paper` to indicate something other than standard US paper (8.5x11 inches) may be useful in some cases.

Your document will be displayed starting at the first page, together with a set of command buttons on the right side of the `xdvi` window. The main keyboard commands, some of which can also be invoked via the command buttons, are described in the following table

Key	Action
q	Quit <code>xdvi</code>
n or f or SPC	Next page
p or b or DEL	Previous page
g	Preceded by page number: Goto page
u, d, l, r	Scroll up, down, left, right
R	Reread <code>.dvi</code> file

You can also use the scrollbars to scroll the screen if necessary. The `R` command allows you to reread the `.dvi` file after re-running  $\text{\LaTeX}$  without exiting and restarting `xdvi`. Very useful.

If the shrink factor is set to any number other than one (either with the command buttons or with the `-s` option), then clicking any mouse button will pop up a “magnifying glass” which shows the unshrunk image in the vicinity of the mouse click. This subwindow disappears when the mouse button is released. Different mouse buttons produce different sized windows. Moving the cursor while holding the button down will move the magnifying glass.

### 3.2 Previewing problems

Of course, things aren’t always so simple, and you may encounter some problems with `xdvi`. This section describes two common ones: fonts and specials.

First, there are the fonts. By default, any fonts installed on our system should be renderable by `xdvi`. If you import a `.dvi` file from another system, `xdvi` may not be able to display it. You would have to obtain the font by other means and tell `xdvi` where to

find it. As described in the manpage `xdvi(1)`, there is an elaborate search mechanism for locating fonts. Basically, you should set the environment variable `XDVIFONTS` to indicate where additional fonts are to be found. An extra colon causes the default directories to be searched. If `XDVIFONTS` isn't set, `xdvi` uses the environment variable `TEXFONTS` used by  $\LaTeX$  instead. You shouldn't need to change these unless you're a  $\LaTeX$  guru, in which case you don't need this guide.

The second issue is so-called “specials.” You will encounter these when you try to use special features such as landscape printing or included figures (described later in this guide). The creator of  $\TeX$ , Donald Knuth, grudgingly admitted that there might be things that  $\TeX$  couldn't do (or at least, couldn't do well), so he added an “escape” mechanism to the  $\TeX$  language. This command, named `\special`, allows information to be placed directly in the `.dvi` file without its being processed by  $\TeX$ . It is assumed that whatever program is printing the `.dvi` file will understand what is meant by this special text, since it is clearly not part of the “normal” formatted output.

This means that often figures or other special formatting aren't correctly rendered by `xdvi`. Viewing your document as Postscript or PDF is generally safer, although sometimes slower.

## 4 Postscript

### 4.1 Postscript files and previewing Postscript

To convert a `.dvi` file to Postscript, use `dvips` with the `-o` option:

```
% dvips paper -P pdf -G0 -o
```

By default the output file will be `paper.ps` if `-o` is the last option in the command, or you can give another filename following the `-o`. When creating Postscript files, it is generally a good idea to additionally specify `-P pdf` to force `dvips` to use outlined rather than bitmapped fonts. The option `-G0` is necessary with some fonts to prevent strange characters in the output, for example for the “fi” ligature.

You can then preview this file using the `gv` or `ghostview` command:

```
% gv paper.ps
```

Since previewing Postscript is guaranteed to be at least as complete as previewing the `.dvi`, why bother with `xdvi`? First, in order to preview with `gv` you have to run `dvips`, which can take time, especially on long documents. Second, and more importantly, `gv` itself is significantly slower than `xdvi`. The reason for this is that whereas the `.dvi` file format is a fairly compact representation of what’s on the page (excluding specials), Postscript is a complete programming language which must be interpreted by the previewer or printer. Thus not only is it slow at first, jumping around between pages often means restarting the interpreter from the beginning of the document. This is not something you want to do to go from page 100 to page 99 of your masterwork.

The moral is to use `xdvi` during the development cycle and learn to use the special features of some of the packages used for including figures described below to minimize the inconvenience. Then use `dvips` with `-o` to convert to Postscript, make a final check with `gv`, then send the Postscript to the printer with `lp` (or save it, or email it, or whatever).

### 4.2 Printing DVI

Calling `dvips` without the `-o` option will send the Postscript output directly to the printer. In its simplest form, you simply invoke it with the basename of your document and it converts the `.dvi` file to Postscript and queues it on the printer:

```
% dvips paper
```

You’ll see output something like the following:

```
This is dvipsk 5.58f Copyright 1986, 1994 Radical Eye Software
' TeX output 1998.08.25:1220' -> !lp -d chaucer -c
<tex.pro>. [1]
request id is chaucer-1147 (standard input)
```

Name	Location	Description
chaucer	737	HP LaserJet IV, 600 dpi
shake	628	HP LaserJet IV, 600 dpi
picasso	737	Tektronix Phaser 560, 600 dpi color laser
dali	737	HP DeskJet 1200C, color (obsolete)
Faculty and Staff use only:		
keats	706	RealTech Laser
poe	707	Apple LaserWriter II

Table 1: Department printers, Fall 1998

What this means is that the document whose “title” is that string with the date and time in it (set by `LATEX` when you formatted the document) is being converted to Postscript and sent to the `lp` program for printing on the printer `chaucer`. It then indicates that it was outputting a “prologue” file (`tex.pro`), followed by the page numbers as they are output, just like `LATEX`. The final line is actually a message from `lp` confirming that the output has been filed for printing.

The printer selected as the destination for `dvips`’ output can be specified several ways. First, if the `-P` option is used, as in

```
% dvips paper -Pshake
```

output will be spooled to that printer. A list of department printers with short descriptions is given in Table 1. Note that different printers have different features and not all are for general use. In particular, the 600 dpi printers produce nicer output faster, and the color printer is much more expensive to use and intended for producing the final version of high-quality transparencies for presentations.

If no `-P` option is given, `dvips` checks the environment variable `PRINTER`. You can set this by adding a line like the following to the `.login` file in your home directory:

```
setenv PRINTER shake
```

Of course, this is only useful if you tend to use one printer primarily.

If you just want to print selected pages from a document, you can use the `-pp` option to `dvips`. For example,

```
% dvips paper -pp 1,5-10
```

will print the first page and the fifth through tenth pages. Note that these page numbers correspond to page numbers, not physical pages. See also the `-p` and `-l` options in the manpage `dvips(1)` for more details on these and the myriad other options.

Style Option	Font Family	rm	it	bf
		tt	sf	sc
avantgarde	AvantGarde	AG Book	AG Book Oblique	AG Demi
		Courier	Helvetica	AG Book SmallCaps
bookman	Bookman	BK Light	BK Light Italic	BK Demi
		Courier	Helvetica	BK Light SmallCaps
chancery	Zapf Chancery	ZC Medium Italic	ZC Medium Italic	ZC Medium Italic
		Courier	Helvetica	ZC Medium Italic
ncs	New Century Schoolbook	NCS	NCS Italic	NCS Bold
		Courier	Helvetica	NCS SmallCaps
palatino	Palatino	Palatino	Palatino Italic	Palatino Bold
		Courier	Helvetica	Palatino SmallCaps
times	Times	Times Roman	Times Italic	Times Bold
		Courier	Helvetica	Times SmallCaps

Table 2: Postscript Fonts

### 4.3 Using Postscript fonts

By default,  $\text{\LaTeX}$  uses the Computer Modern family of fonts designed by Donald Knuth using METAFONT. It uses what it calls `tfm` ( $\text{\TeX}$  font metric) files to determine the size of each character, and then programs that process DVI files (such as `xdvi` or `dvips`) use what they call `pk` files to actually render the image of the character. There are two reasons why you might want to use something else, in particular fonts described using the Postscript language.

First, you generally end up with smaller Postscript files, since Postscript fonts are usually defined internally by the Postscript printer. With Computer Modern fonts, a large part of the Postscript file created by `dvips` is in fact Postscript code that defines the bitmap images of all the characters in any font you use. This can easily be 80% of the size of the file, or more for short documents. By using Postscript fonts, you no longer need this information to be passed to the printer, resulting in smaller files, less disk usage, quicker transfer to the printer (or via email or whatever), and faster printing.

Second, you might prefer the the look of the Postscript fonts. For example, Times is popular because it is a very dense typeface relative to Computer Modern and hence can fit more text into the same number of pages (good for papers with page limits). Helvetica is popular with Macintosh users, as is New Century Schoolbook. The list goes on an on. There are tons of Postscript fonts out there—perhaps your next paper could be in Zapf Dingbats.

To use Postscript fonts for a document rather than Computer Modern, use the appropriate package (using `\usepackage`) from the list shown in Table 2. The table shows which font is selected by the various  $\text{\LaTeX}$  type style commands. All of these styles continue to use Computer Modern Math Italic for math mode, which is what it was designed for, after all.

## 5 Creating PDF

Postscript can be converted to Adobe PDF using `ps2pdf`.

## 6 Creating and Including Figures

$\LaTeX$  (and  $\TeX$  on which it is based) were designed to format text elegantly, in particular mathematical equations. That’s fine for the theory folks, but these days, with graphical user interfaces so important, and with the research in the department on vision-related projects, and even the age-old need to display statistical data effectively, we need additional tools for creating figures and adding them to  $\LaTeX$  documents. This section describes the range of options and some of the tools used with them.

### 6.1 $\LaTeX$ picture mode

The simplest way of generating figures is  $\LaTeX$ ’s own `picture` environment, described in The  $\LaTeX$  Book. This lets you draw “block diagram” figures as if they were laid out on graph paper and are directly interpreted by  $\LaTeX$ . This means that the resulting figures can be previewed by `xdvi`. It also means that all of  $\LaTeX$ ’s considerable power at formatting equations, using fonts, and the like can be employed in your figure. While picture mode’s graphics are quite limited, and it is difficult to lay them out by hand, if your figure can be described using them, it is possible to use a tool like `xfig` (described in Section 6.3.1) to generate the  $\LaTeX$  picture mode commands.

A very powerful set of commands for creating diagrams in  $\LaTeX$  is provided by the `pstricks` package. Because this package uses Postscript commands, it must be used with plain  $\LaTeX$  rather than `pdf $\LaTeX$` . Making diagrams with `pstricks` is kind of like programming a drawing – it can be much more time consuming than using a WYSIWYG drawing program, but can be useful for creating figures automatically from program output.

A useful package for producing tree diagrams is `pst-qtrees` by David Chiang. This package is based on `pstricks` and therefore does not work with `pdf $\LaTeX$` .

### 6.2 Postscript and includegraphics

At the other extreme from picture mode is the use of Postscript (mentioned in Section 4) to describe the figure. Postscript figures have none of the limitations of picture mode and can describe almost arbitrarily complicated figures, including color and greyscale images. The problem is that they are not part of the  $\LaTeX$  language, and thus can only be used via the `\special` command mentioned in Section 3.2. They sometimes are not rendered correctly when previewing using `xdvi`. It would also be very hard to create complicated Postscript figures by hand, but luckily most drawing programs on many platforms will output Postscript, so we don’t have to worry about that. Some of these tools are described in the next section.

In order to include a Postscript figure in a  $\LaTeX$  document, you need to do two things. First, you must use a `\usepackage` command in the preamble of your document (after `\documentclass` but before `\begin{document}`). Both the `graphics` and `graphicx` (“extended graphics”) packages are standard in  $\LaTeX 2_{\epsilon}$ . The `graphics` package is described in the  $\LaTeX$  book (see Section 11). However, I recommend using the more understandable `graphicx` package by adding the following line to your preamble:

```
\usepackage{graphicx}
```

Then, at the point where you want to include the Postscript figure, use the `\includegraphics` command. The basic usage would be, for example:

```
\includegraphics{diagram.eps}
```

where `diagram.eps` is the file we want to include.

PDF $\LaTeX$  can include PDF files, but not Postscript. If you leave out the file name extension in the `\includegraphics` command, you can compile your document with either plain  $\LaTeX$  (which will look for a `.eps` file) or PDF $\LaTeX$  (which will look for a `.pdf` file).

Typically you won't want a figure just in isolation like this, unless it is intended as a special character or other inline drawing. A more typical usage, which you can use as a guide for your own figures is the following:

```
\begin{figure}
  \begin{center}
    \includegraphics{diagram.eps}
    \caption{This is a sample figure}
    \label{f:sample-figure}
  \end{center}
\end{figure}
```

Instead of the `center` environment, you can use the command `\centering`, which doesn't add any vertical space. There are plenty of other possibilities.

Before going on, we need a quick digression on terminology. While any Postscript file can be included in a document with `includegraphics`,  $\LaTeX$  needs to know how much space to put aside for the figure, which it will treat as a single box. An *encapsulated* Postscript file includes this information in the file in the form of a comment like

```
%%BoundingBox: 11x 11y urx ury
```

where `(11x,11y)` are the coordinates of the lower, left corner of the figure's "bounding box," and `(urx,ury)` the coordinates of the upper, right corner, in units of one seventy-second of an inch. Encapsulated Postscript files are conventionally named ending in `.eps`. Most programs should allow you to save your figure as Encapsulated Postscript. If your Postscript file does not have a bounding box comment, you can either add one by hand or specify it as an option to the `\includegraphics` command, as in

```
\includegraphics[bb=0 0 W H]{diagram.eps}
```

In this example, you can use the width (`W`) and height (`H`) of the figure for the bounding box.

Assuming you have a Postscript figure with a bounding box (or you can add the bounding as described above), you can stretch or rotate the figure using options to `\includegraphics`. Examples are:

```

\includegraphics[height=1in]{diagram.eps}
\includegraphics[height=1in,width=3in]{diagram.eps}
\includegraphics[angle=90]{diagram.eps}

```

The first example scales in both directions equally so that the resulting height is one inch. The second example scales (possibly unevenly) to the desired size, and the third example rotates ninety degrees counterclockwise.

The final feature of `includegraphics` worth mentioning here is draft mode. If you specify the `draft` option when including the `graphicx` package, subsequent figures will be replaced by an empty box containing the name of the file that would be included. This can be significantly faster to format or print, results in much smaller Postscript files, and, most importantly, can be previewed by `xdvi` without complaint.

More details on `graphicx`, including how to use it with compressed Postscript files, can be found in the document in the “Miscellaneous Manuals” binders in the lab.

## 6.3 Tools for creating figures

This section provides very rudimentary descriptions of some of the most popular tools used in the department for creating figures. The emphasis is on getting their output into  $\LaTeX$  documents. For more details, see their manpages or manuals in the lab.

### 6.3.1 Xfig: Object-oriented diagrams

The `xfig` program is an object-oriented figure editor that is especially good for block diagrams, flowcharts, and the like. It supports color, layered objects, objects connected by links that adjust with the objects, Postscript fonts, and much more. The interface takes some getting used to, but is fairly consistent once you master it. The manpage provides a detailed summary of functionality and options.

The strongest point in favor of `xfig` is that it can output in many  $\LaTeX$ -friendly file formats, via the program `fig2dev` invoked by the “Export” button on the `xfig` screen. It can generate  $\LaTeX$  picture mode directly, for figures that use only those features, and can constrain lines and arrows to acceptable slopes for picture mode. Mark text as “special” and you can have it processed by  $\LaTeX$  within the figure, which is great for state diagrams and the like. It also generates Postscript and Encapsulated Postscript for inclusion using `includegraphics`.

For figures that are too complex for picture mode but require  $\LaTeX$  formatting of equations or the like, `fig2dev` supports the `pstex` and `pstex_t` formats. The former outputs everything *except* text marked special to an encapsulated Postscript file. The latter writes *only* text marked special to a  $\LaTeX$  picture mode file that positions the text appropriately. You then include *both* files using something like the following, assuming the files are `figure.eps` and `figure.tex`:

```

\begin{picture}(0,0)%
\includegraphics{figure.eps}%

```

```
\end{picture}%  
\input{figure}
```

The % characters indicate a comment to T<sub>E</sub>X and, in this case, prevent the newlines from introducing space that would cause the figures to not be perfectly overlaid.

Another possibility for combining graphics and T<sub>E</sub>X formatting of text is the `psfrag` package, and its associated program `ps2frag`. If you're interested in these, you can find them in the installed L<sup>A</sup>T<sub>E</sub>X directories.

### 6.3.2 Dia: Free-hand drawing

The `dia` program provides similar wysiwyg functionality to `xfig`, but has a much more friendly and modern interface. Starting the program pops up the tool palette, from which you can start a new figure. You can save in Postscript format, or to a MetaPost file, which has the benefit of being able to include T<sub>E</sub>X formatting or formulas in the labels of your diagram.

To convert from `dia` to MetaPost:

```
% dia diagram.dia --export=diagram.mp  
diagram.dia --> diagram.mp
```

Then run MetaPost:

```
% mpost diagram.mp  
This is MetaPost, Version 0.641 (Web2C 7.5.4)  
(diagram.mp [1] )  
1 output file written: diagram.1  
Transcript written on diagram.log.
```

In your L<sup>A</sup>T<sub>E</sub>X file:

```
\usepackage{graphicx}  
...  
\includegraphics{diagram.1}
```

### 6.3.3 GIMP: GNU Image Manipulation Program

The `gimp` is basically a Photoshop clone, but distributed as free software and running on Unix machines. It combines image manipulation with direct painting, and supports the now-standard “plug-in” approach to providing zillions of whizzy (and generally useless) effects. Highly recommended, although the version current as of this writing is not overly stable. Save early and save often.

`gimp` can be used to capture screen snapshots.

### 6.3.4 ImageMagick: General image processing

The `convert` command will convert between almost any two image formats, and is useful for converting images to Postscript files suitable for inclusion using `includegraphics`.

You should note that including images using Postscript is done by literally including an ASCII representation of your raster array in the Postscript file, together with a small bit of Postscript code for drawing it on the page. As a result, the Postscript file will be large, sometimes very large, sometimes very, very large. This will result in slow previewing and printing and sometimes even “out of memory” errors in the printer. You have been warned.

### 6.3.5 Gnuplot: General purpose plotting

The `gnuplot` package plots mathematical functions and statistical data in two or three dimensions and displays the results on your screen. It permits detailed control of the resulting plots and can write files in a variety of formats. Online hierarchical help is available by typing `help` at the `gnuplot` prompt.

The following will change the format of the output (from the default, `x11`, which displays on the screen) and redirect the output into a file:

```
set output "plot.eps"
set terminal postscript
```

the result is an Encapsulated Postscript file suitable for inclusion using `includegraphics`. To return output to the terminal, use

```
set output
set terminal x11
```

You can also select terminal type `latex`, which will output picture mode commands, but be careful. The output will have many, many individual points being plotted. The result can easily overwhelm  $\text{\LaTeX}$  or take a very long time to format, preview, and print.

### 6.3.6 Maxima: Interactive mathematical computation

The `maxima` package is an interactive system for symbolic algebra. It calls `gnuplot` for its plotting functions, and can be used directly to generate postscript plot by specifying the option `[GNUPLLOT_TERM, PS]` to the `plot2d` or `plot3d` commands:

```
% maxima
(%i2) plot3d (y*sin (x) , [x,-%pi,%pi],[y,-1,1],[grid,50,15],
            [GNUPLLOT_TERM, PS], [GNUPLLOT_OUT_FILE, "myplot.ps"]);
```

### 6.3.7 R: Statistical Computing

The package R is an open-source version of Splus, and is useful for statistical tasks such as regression and much more. To make a simple scatter plot with R:

```
% R
> x <- c(0.02, 0.02, 0.06, 0.06, 0.11, 0.11, 0.22, 0.22, 0.56, 0.56, 1.10, 1.10)
> y <- c(76, 47, 97, 107, 123, 139, 159, 152, 191, 201, 207, 200)
> postscript("aplot.ps")
> plot(x,y)
```

### 6.3.8 Matlab: Matrix laboratory

**From the manpage matlab(1):**

Matlab is an interactive “matrix laboratory” for tasks involving matrices, graphics and general numerical computation. It reads commands from the standard input, as well as from files known as M-files, ASCII files with an extension of `.m`. Collections of M-files which extend matlab’s applicability to specialized fields are known as toolboxes.

Matlab is also a scientific programming language, with some of the elements of languages like Algol, Fortran, Pascal and C. One important distinction is that the only data type in matlab is the matrix, a rectangular array of possibly complex floating point values, which need not be dimensioned, and whose size and shape can vary dynamically. In some situations, special meaning is attached to n-by-1 matrices, columns, to 1-by-n matrices, rows, to 1-by-1 matrices, scalars, to matrices of ASCII characters, strings, and even to empty 0-by-0 matrices.

**From Mike Marchetti (mwm):**

To generate Matlab output suitable for inclusion in L<sup>A</sup>T<sub>E</sub>X, do the following in Matlab:

```
print -deps filename.eps
```

for black-and-white printers, or

```
print -depsc filename.eps
```

for color printers.

### 6.3.9 OpenOffice suite

The OpenOffice set of tools offers the most “Microsoft-like” applications available on department workstations, including spreadsheet, drawing, and presentation programs.

You can save in Encapsulated Postscript, which can then be included in your document using `includegraphics`.

## 7 Checking your spelling

Perhaps this section should be right after the basics of using L<sup>A</sup>T<sub>E</sub>X, because the fact is that any document should be spellchecked early and often to avoid typos. You have several possibilities for checking the spelling of your document.

The most basic is the Unix `spell` program. Run the command

```
% spell paper.tex
```

It will print a list of misspelled words to the standard output, one per line. Unfortunately, this will include many L<sup>A</sup>T<sub>E</sub>X commands that don't correspond to English words. You can instead use

```
% detex paper.tex | spell
```

The `detex` program strips most T<sub>E</sub>X and L<sup>A</sup>T<sub>E</sub>X constructs, and expands `\input` and `\include` commands (unless the `-i` option is given). More details on both of these are available from their manpages. The `spell` program is of limited use because you have to then go through manually and correct any misspellings.

A better spelling checker is the `ispell` program, which uses a better dictionary, provides an interactive, screen-oriented display, and allows you to add words to a personal dictionary. Run the command

```
% ispell paper.tex
```

Then `ispell` will display each word that does not appear in the dictionary at the top of the screen and allow you to change it. If there are “near misses” in the dictionary (words which differ by only a single letter, a missing or extra letter, a pair of transposed letters, or a missing space or hyphen), then they are also displayed on following lines. As well as “near misses,” `ispell` may display other guesses at ways to make the word from a known root, with each guess preceded by question marks. Finally, the line containing the word and the previous line are printed at the bottom of the screen. If your terminal can display in reverse video, the word itself is highlighted. You have the option of replacing the word completely, or choosing one of the suggested words. Type `?` to see the help screen detailing the possible commands. The manpage provides extensive detail of the commands, options, and dictionary techniques used by `ispell`.

Finally, if you use the Emacs editor (and why wouldn't you, given the availability of AUC-T<sub>E</sub>X, described in Section 10), you can run `ispell` from within Emacs. Just do

```
M-x ispell-buffer RET
```

to run `ispell` on the contents of the current buffer. You get the same highlighting, alternatives, and commands as basic `ispell`, but within Emacs. You can also use `ispell-word` to check the spelling of the word under point. To replace Emacs' built-in word spelling checker, add the following to the `.emacs` file in your home directory:

```
(global-set-key "\e$" 'ispell-word)
```

Then, to spellcheck the word under point, do `M- $\$$` .



## 8 Using `BIBTEX` for bibliographic citations

Since the main use of `LATEX` in the department is the production of academic papers, and since, as we all know, the key to successful academic writing is citing other people’s work, `LATEX` has special features to deal with bibliographic citations. Much of this is described in the `LATEX` book’s description of `BIBTEX`, and I’m not going to repeat it here. This section provides a quick review of the basic concepts and how to use `BIBTEX` on our system.

Looking at the right side of the flowchart in Figure 1, you can see that `BIBTEX` uses information written by `LATEX` to the `.aux` file, combines it with the bibliographic database files (`.bib`) and generates the bibliography file (`.bbl`). The `.blg` file contains a log of what was done, similar to `.log`.

In more detail, each time you use the `\cite` command in your `.tex` file(s), `LATEX` writes an entry to the `.aux` file. Similarly, the `\bibliographystyle` and `\bibliography` commands are recorded in the `.aux` file. When you run `BIBTEX`, it scans the `.aux` file and attempts to find cited items in the specified bibliographic database files. It then puts together a citation formatted according to the requested bibliography style and outputs it to the `.bbl` file. The next time `LATEX` is run, when it encounters the `\bibliography` command it also reads in the `.bbl` file, thereby adding the list of references to the document.

This is similar to the multi-pass processing of the `.aux` file described in Section 2.4. In particular, note that running `BIBTEX` can generate errors that are not noticed until the next time `LATEX` is run. If the entry in the `.bib` file contains a `LATEX` error, `BIBTEX` will blindly copy it to the `.bbl` file. Then, when `LATEX` is next run and the `.bbl` file is read in, the error will be signaled. The solution, as with `.aux` file errors, is to note carefully which file is being read when the error occurred. If it was the `.bbl` file, the problem is almost certainly in your `.bib` file, which should be corrected, `BIBTEX` re-run, then `LATEX` re-run at least once.

Here is one more tip that may be useful in using `BIBTEX` effectively. It is often the case that by the time you have your document ready, it’s still just a little bit over the page limit for wherever you’re sending it. Short of deleting some of your precious text, what can you do? Among other tricks, there’s nothing preventing you from editing the `.bbl` file to shorten it somewhat—it’s just regular, albeit somewhat cryptic, `LATEX` input. Now, the “correct” thing to do would be to find a bibliography style that abbreviates properly, or whatever, but in practice it’s often easier to just go in and shorten some of those conference titles by hand. Of course, then next time you run `BIBTEX` you’ll recreate the `.bbl` file and lose your changes, but this is intended as a last resort after all the citations are in.

More details on `BIBTEX`, including changes from the description in the `LATEX` and a primer on creating `BIBTEX` styles, can be found in the document in the “Miscellaneous Manuals” binders in the lab.



## 9 Preparing slides for presentations

Another thing we tend to do a lot of in the department is talk about our work, and slides are the mainstay of computer science talks. The details of using the `slides` document class to produce slides are in *The L<sup>A</sup>T<sub>E</sub>X Book* – one useful L<sup>A</sup>T<sub>E</sub>X package for producing nifty-looking slides is `beamer`.

A few alternatives that people have used in the department to produce slides:

First, some people like the logical layout capabilities of `xfig` (described in Section 6.3.1) for drawing slides, combined with the ability to use L<sup>A</sup>T<sub>E</sub>X fonts and formatting (using `pstex` and `pstex_t` output). It's not very good at keeping track of collections of slides though.

The ultimate tool for slides is `OpenOffice`, part of the suite described in Section 6.3.9. Since it has nothing to do with L<sup>A</sup>T<sub>E</sub>X, I won't go into any more details, except to note that it makes very whizzy color slides that blow the socks off `slides` slides.

And finally, there are versions of the Microsoft Office suite, specifically PowerPoint, on both the public Macs and NT workstations. The drawback of these is that they have nothing to do with Unix (or is that a feature...).



## 10 Editing L<sup>A</sup>T<sub>E</sub>X documents with AUC-T<sub>E</sub>X

As you well know, L<sup>A</sup>T<sub>E</sub>X is not a WYSIWYG formatter. In fact, it has a verbose and sometimes arcane command language that can often be tedious, at best, to work with. However, a package named AUC-T<sub>E</sub>X for the Emacs editor is available to make life easier. In addition to providing convenient keystroke commands for common L<sup>A</sup>T<sub>E</sub>X commands, it allows you to edit, format, preview, and print your documents all from within Emacs.

This section describes AUC-T<sub>E</sub>X briefly. It assumes you know how to use Emacs, if not how to program it. More information is available from the Info help pages available from Emacs' online help facility. In describing keybindings, the description "C-x" means "control-x," *i.e.*, hold down the Control key and type x, M-x means "meta-x," RET stands for the Return key, and TAB for the Tab key. Where appropriate, a function name is given after a keybinding so you can look up the function for more details.

### 10.1 Accessing AUC-T<sub>E</sub>X

To use AUC-T<sub>E</sub>X, you need only add the following to the `.emacs` file in your home directory:

```
(autoload 'tex-mode "auctex" "Improved TeX/LaTeX mode" t)
(autoload 'latex-mode "auctex" "Improved TeX/LaTeX mode" t)
```

Then, when you visit a file whose name ends in `.tex`, Emacs will automatically load the AUC-T<sub>E</sub>X package and put the buffer visiting that file in the appropriate mode. You will notice messages as Emacs loads the files the first time you use the mode.

AUC-T<sub>E</sub>X is intended to make constructs used commonly in T<sub>E</sub>X and L<sup>A</sup>T<sub>E</sub>X documents easier to type by setting up keybindings that insert them.

#### 10.1.1 Font Changes

Font changing commands are inserted by typing C-c C-f (T<sub>E</sub>X-font) followed by a key for the desired font:

Key Sequence	Result
C-c C-f C-b	<code>\textbf{}</code>
C-c C-f C-c	<code>\textsc{}</code>
C-c C-f C-e	<code>\emph{}</code>
C-c C-f C-i	<code>\textit{}</code>
C-c C-f C-r	<code>\textrm{}</code>
C-c C-f C-s	<code>\textsl{}</code>
C-c C-f C-t	<code>\texttt{}</code>

AUC-T<sub>E</sub>X positions point at the appropriate place inside the parentheses.

### 10.1.2 Environments and Sections

Environments (*i.e.*, constructs starting with `\begin{...}` and ending with `\end{...}`) are inserted by typing `C-c C-e` (`LaTeX-environment`). You are prompted for the type of environment (*e.g.*, `itemize`, hit `TAB` for completions). Some environments prompt for additional parameters, such as the `table` or `figure` environments that prompt for position of the figure, the caption, the label, and whether centering is desired. Within a list environment, `M-RET` (`LaTeX-insert-item`) will insert a newline and the appropriate `\item` command for the environment. You can close an open environment with `C-c ]` (`LaTeX-close-environment`).

Sectioning commands (*e.g.*, `\chapter`, `\section`, *etc.*) are inserted by typing `C-c C-s` (`LaTeX-section`). You are prompted for the level of the section (again, hit `TAB` for completions) and a label to be used in a `\label` command (just hit `RET` to not have a label).

### 10.1.3 Formatting, Previewing, Printing

To format, preview, or print the document, type `C-c C-c` (`TeX-command-master`). You are prompted for what operation you want to perform (hit `TAB` for completions). The following are some of the possible commands and what they do:

Key Sequence	Result
<code>C-c C-c LaTeX RET</code>	Run <code>L<sup>A</sup>T<sub>E</sub>X</code>
<code>C-c C-c BibTeX RET</code>	Run <code>B<sub>I</sub>B<sub>T</sub>E<sub>X</sub></code>
<code>C-c C-c Spell RET</code>	Spellcheck document
<code>C-c C-c Check RET</code>	Check document with <code>lacheck</code>
<code>C-c C-c View RET</code>	View DVI file with <code>xdvi</code>
<code>C-c C-c File RET</code>	Create Postscript file with <code>dvips</code>
<code>C-c C-c Print RET</code>	Print document via <code>dvips</code>

You need only type the first letter of the operation, for example “l” to run `LATEX`, then hit `RET`.

Details of the formatting are displayed in another buffer. If there are errors, you can type `C-c ‘` (`TeX-next-error`) to locate the next error, position your buffer at the appropriate place in the input file, and display helpful text about the error.

One other feature is worth mentioning. If your document is split into multiple source (`.tex`) files, it can be annoying to remember to switch buffers into the “master” file before typing `C-c C-c`. If you forget, `LATEX` will scream about your ill-formed input since it didn’t get the master file information. The solution is to start all “subsidiary” files with a line

```
%% Master: paper.tex
```

where `paper.tex` is the name of your master file. The processing commands will use that file rather than the file being edited in the current buffer. `C-c ‘` understands about multiple files also, which is nice.

## 10.2 Other features

There are many other nice features of AUC-TeX, such as the ability to do outlining based on L<sup>A</sup>T<sub>E</sub>X sectioning commands, format parts of documents, and special commands for math mode. For more details on these you are referred to the Info documents available from Emacs' online help facility. You can access it by doing `M-x info RET`, then selecting the topic "AUC-TeX" using the mouse or by typing it after typing `m` (for "menu").



## 11 Further Reading

The definitive reference for T<sub>E</sub>X is *The T<sub>E</sub>Xbook*, by Donald Knuth (Addison Wesley, 1984, ISBN 0-201-13447-0, paperback 0-201-13448-9).

For L<sup>A</sup>T<sub>E</sub>X (that is, L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub>), see *L<sup>A</sup>T<sub>E</sub>X, a Document Preparation System, 2nd ed.*, by Leslie Lamport (Addison Wesley, 1994, ISBN 0-201-52983-1).

The University of Rochester Computer Science Department styles and macros are described in Department Guide number 13. These work with L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> provided you make the appropriate changes from `\documentstyle` to `\documentclass` and use `\usepackage` for the department packages.

As of this writing, the main repository for T<sub>E</sub>X and L<sup>A</sup>T<sub>E</sub>X macros and programs (including those programs themselves) is `ftp.shsu.edu`, part of the Comprehensive T<sub>E</sub>X Archive Network (CTAN).

The Frequently-Asked-Questions list for T<sub>E</sub>X is posted regularly to the newsgroups `news.answers` and `comp.text.tex`. It and the “Supplementary T<sub>E</sub>X Information” list provides answers to common questions and pointers to further information. It (and all FAQ lists) are archived at `rtfm.mit.edu` from where they can be retrieved via anonymous FTP.



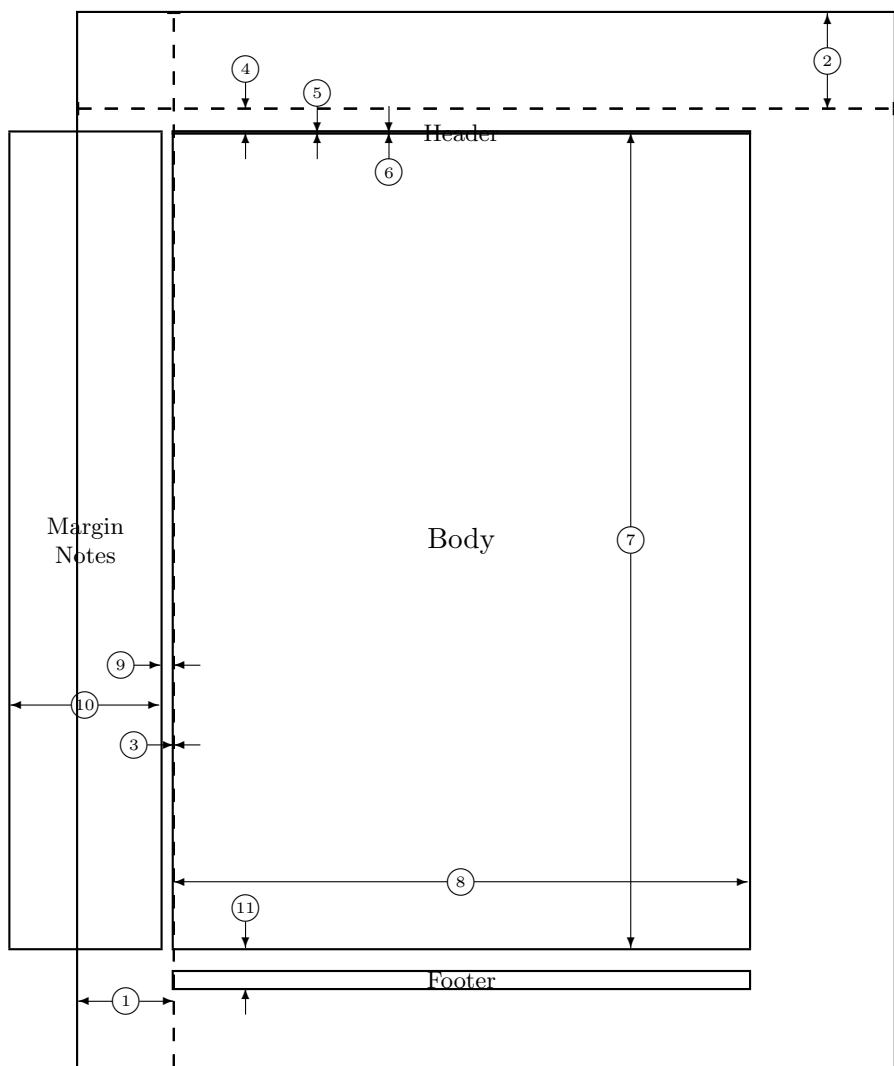
## A The $\LaTeX$ Local Guide

In several places in the  $\LaTeX$  book, you are referred to your “local guide” for details that vary from site to site. Some of the material in that document is either outdated or is superseded by the descriptions given here. This section contains all the material from Leslie Lamport’s original “local guide” (`local.tex`) as modified by Ken Yap and Lawrence Crowl (`urcslocal.tex`) that I deemed didn’t suffer from either of the above conditions. It’s probably still confusing, but at least the information’s here.

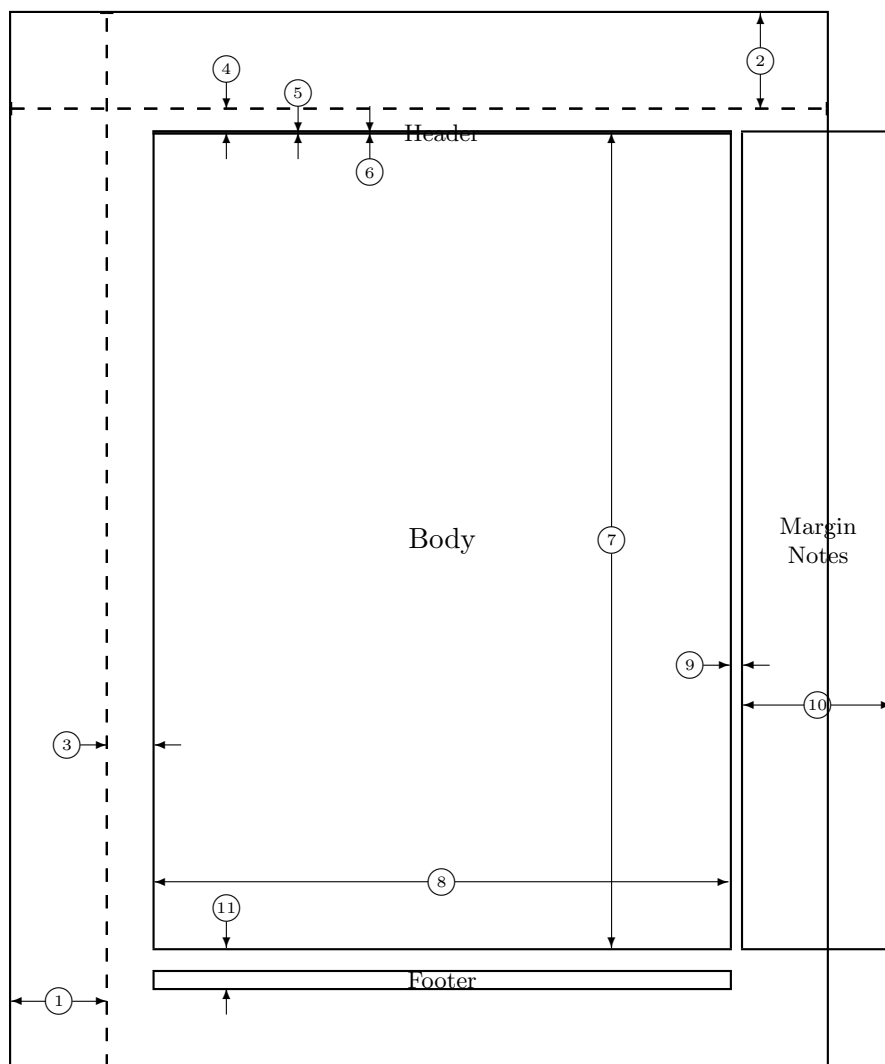
This section has not been updated for  $\LaTeX 2_{\epsilon}$ .

### A.1 Layout Parameters

There are many parameters which control the layout of pages. The following two pages illustrate the page layout and list the parameters. To experiment with different settings and see the effect on the figures, use the `layout` package, modify the parameters, then invoke `\layout` to create the figure.



- |    |                       |    |                                  |
|----|-----------------------|----|----------------------------------|
| 1  | one inch + \hoffset   | 2  | one inch + \voffset              |
| 3  | \evensidemargin = 0pt | 4  | \topmargin = 18pt                |
| 5  | \headheight = 0pt     | 6  | \headsep = 0pt                   |
| 7  | \textheight = 614pt   | 8  | \textwidth = 433pt               |
| 9  | \marginparsep = 10pt  | 10 | \marginparwidth = 113pt          |
| 11 | \footskip = 30pt      |    | \marginparpush = 5pt (not shown) |
|    | \hoffset = 0pt        |    | \voffset = 0pt                   |
|    | \paperwidth = 614pt   |    | \paperheight = 794pt             |



- |    |                       |    |                                  |
|----|-----------------------|----|----------------------------------|
| 1  | one inch + \hoffset   | 2  | one inch + \voffset              |
| 3  | \oddsidemargin = 36pt | 4  | \topmargin = 18pt                |
| 5  | \headheight = 0pt     | 6  | \headsep = 0pt                   |
| 7  | \textheight = 614pt   | 8  | \textwidth = 433pt               |
| 9  | \marginparsep = 10pt  | 10 | \marginparwidth = 113pt          |
| 11 | \footskip = 30pt      |    | \marginparpush = 5pt (not shown) |
|    | \hoffset = 0pt        |    | \voffset = 0pt                   |
|    | \paperwidth = 614pt   |    | \paperheight = 794pt             |

## A.2 Document Styles

This section can be skipped at first reading.

There are six document styles and style options available here that are not described in the manual: the `proc` style option for making camera-ready copy for conference proceedings, the `doublespace` style option for double spacing, the `bezier` option for drawing curves, the `ifthen` option for implementing **if-then-else** and **while-do** control structures, the `letter` style for making letters, the `showidx` option for printing index entries in the margin, and the `makeidx` option for use with the `makeindex` program. See Section A.4 for further information on the *MakeIndex* program that uses the `makeidx` option. The remaining style options are described below.

### A.2.1 The `proc` Style Option

The `proc` option is used with the `article` document style. It produces two-column output for ACM and IEEE conference proceedings. The command `\copyrightspace` makes the blank space at the bottom of the first column of the first page, where the proceedings editor will insert a copyright notice. This command works by producing a blank footnote, so it is placed in the text of the first column. It must go after any `\footnote` command that generates a footnote in that column.

$\text{\LaTeX}$  automatically numbers the output pages. It's a good idea to identify the paper on each page of output. Placing the command

```
\markright{Jones---Foo}
```

in the preamble (before the `\begin{document}` command) prints “Jones—Foo” at the bottom of each page.

For final, camera-ready copy, publishers often want the manuscript without page numbers or headings. In this case, leave off the `\markright` command and place

```
\pagestyle{empty}
```

before `\begin{document}`.

### A.2.2 Double Spacing

Conference submissions and theses may require double spaced text. For this, include the `doublespace` style option that turns on double spacing and defines the `singlespace` environment. Place the `singlespace` environment around any figures, tables, quotations, programs, *etc.* that will look best single spaced.

The `doublespace` option stretches lines by a factor of two. If this is too much, put the command `\setstretch{1.7}` (or some appropriate value) before the `\begin{document}` command.

For example,

```

\documentstyle[doublespace]{article}
\setstretch{1.5}
\begin{document}
Most of the document will be double spaced text.
\begin{singlespace}
Some single spaced text.
\end{singlespace}
\end{document}

```

Sometimes a document is to appear in both single spaced and double spaced form. To format the document for single spacing, simply use `\setstretch{1}`.

### A.2.3 The bezier Style Option

This option defines a single command, `\bezier`, that draws a curved line in a `picture` environment. I can't believe that anyone would do this now that there are the tools described in Section 6.3. So I've deleted the description.

### A.2.4 The ifthen Style Option

This option provides two programming language features that are useful only for people who already know how to program. It defines the two commands

```

\ifthenelse{test}{then clause}{else clause}
\whiledo{test}{do clause}

```

that implement the following two Pascal language structures

```

if test then then clause
    else else clause
while test do do clause

```

The *then*, *else*, and *do* clauses are ordinary L<sup>A</sup>T<sub>E</sub>X input; *test* is one of the following:

- A relation between two numbers formed with `<`, `>`, or `=`; for example, `\value{page}>3`.
- `\equal{string1}{string2}`, which evaluates to *true* if *string1* and *string2* are the same strings of characters after all commands have been replaced by their definitions. (Upper- and lowercase letters are unequal.)
- A logical combination of the above two kinds of tests using the operators `\or`, `\and`, and `\not` and the parentheses `\(` and `\)`—for example:

```

\not \( \value{section} = 1 \and \equal{Jones}{\myname} \)

```

The *test* argument is a violently moving argument, which means that not only fragile commands but even some commands that are not normally fragile will break, causing T<sub>E</sub>X to enter an infinite loop. The `\protect` command works in these situations.

These commands, together with `\renewcommand` and the commands of Section C.7.4 for manipulating counters, open up a whole new world of hacking.

### A.2.5 Letters

The `letter` document style, described in the manual, should be used for generating personal letters. Mailing labels are formatted in two columns of five  $2'' \times 4\text{-}1/4''$  labels each, suitable for copying onto Avery brand, number 5352 address labels.

There is a style, `urcsletter`, for generating letters on Department letterhead. It is described in the “Guide to Department Styles,” Department Guide 13.

### A.2.6 The `showidx` Style Option

This style option, for use with the `report` or `book` document styles, causes index entries to be printed in the outer margin. It does not change the effect of `\makeindex`, which controls whether or not an `.idx` file is written. No attempt is made to avoid overprinting marginal notes. This option issues a `\flushbottom` declaration.

## A.3 Where the Files Are

All  $\text{\LaTeX}$  files mentioned in the manual, including the `sty` and `doc` files, are in the directory `/usr/staff/lib/tex/inputs`, in a subdirectory that depends on what package they came from. The `tfm` files used by  $\text{\TeX}$  and the `pk` (raster) files used by `xdvi` and `dvips` are in `/usr/staff/lib/tex/fonts`.

## A.4 Makeindex: An index processor for $\text{\LaTeX}$

This section contains the text of Leslie Lamport’s document of the same name, dated February 17, 1987.

### A.4.1 How to Use *MakeIndex*

*MakeIndex* is a program for making an index in a document generated with  $\text{\LaTeX}$ . The first step in producing the index is to put the necessary `\index` commands in your document, as described in the next section. Here, I describe how to generate the index after the `\index` commands are in place.

Let’s suppose that the root file of your document is `myfile.tex`. You must make the following changes to your document:

- Add the `makeidx` document-style option to the list of options in the `\documentstyle` command. (See page 21 of the  $\text{\LaTeX}$  manual.)
- Put a `\makeindex` command in the preamble (between the `\documentstyle` and `\begin{document}` commands).
- Put a `\printindex` command where you want the index to appear—usually at the end, right before the `\end{document}` command.

You then run  $\text{\LaTeX}$  on your entire document, causing it to generate the file `myfile.idx`, which I will call the `idx` file. Next, run the *MakeIndex* program by typing the following Unix command:

```
makeindex myfile.idx
```

This produces the file `myfile.ind`, which I will call the `ind` file. If *MakeIndex* generated no error messages, you can now rerun  $\text{\LaTeX}$  on your document and the index will appear. (You can remove the `\makeindex` command first so the `idx` file is not regenerated.) If there were error messages, see Section A.4.3.

By reading the index, you may discover additional mistakes. These should be corrected by changing the appropriate `\index` commands in the document and regenerating the `ind` file. If there are problems that cannot be corrected in this way, you can always edit the `ind` file directly. However, such editing is to be avoided because it must be repeated every time you generate a new version of the index.

## A.4.2 How to Generate Index Entries

**A.4.2.1 When, Why, What, and How to Index** It's tempting to generate the index as you write the document. Resist the temptation. It is virtually impossible to obtain any consistency in an index that is generated in this way.

An index is there to help the reader find what he's looking for. With this in mind, common sense can help in figuring out what should be in the index and how it should be organized. Since it's often hard to distinguish common sense from equally common nonsense, professional advice is useful. Many style guides discuss indexing; the pamphlet *Indexing Your Book* by Sina Spiker (The University of Wisconsin Press, 1954) is, according to its subtitle, "A Practical Guide for Authors".

Unfortunately, these guides to indexing seem to have been written when high tech meant using a ball-point pen instead of a quill, so their advice on the mechanics of creating an index revolve around how to stack your  $3 \times 5$  index cards. You'll have to figure out your own method of using the computer to lighten the chore. An alphabetical list of every word in your document, with duplicates removed, is not a bad place to start. If your system has Howard Trickey's `delatex` program, then the following Unix command generates such a list from the file `myfile.tex` and puts it on the file `foo`:

```
delatex myfile.tex | sort -uf > foo
```

The computer is only a tool; it can't write the index for you. It may be easy to choose which words are important and mechanically generate an index citing every occurrence of those words, but the resulting index will not be as useful to the reader as one prepared with more care.

**A.4.2.2 The Basics** Each `\index` command causes  $\text{\LaTeX}$  to write an entry on the `idx` file. The following example shows some simple `\index` commands and the index entries that

they produce. The page number refers to the page containing the text where the `\index` command appears.<sup>2</sup>

Page ii	<code>\index{Alpha}</code>	Alpha, ii
Page viii:	<code>\index{alpha}</code>	alpha, viii, ix, 22
Page ix:	<code>\index{alpha}</code> <code>\index{Alphabet}</code>	alpha bet, 24 Alphabet, ix
Page 22:	<code>\index{alpha}</code> <code>\index{alphas}</code>	alphabet, 23 alphas, 22
Page 23:	<code>\index{alphabet}</code> <code>\index{alphabet}</code>	
Page 24:	<code>\index{alpha bet}</code>	

Note that the duplicate `\index{alphabet}` commands on page 23 produce only one “23” in the index.

To produce a subentry, the argument of the `\index` command should contain both the main entry and the subentry, separated by a `!` character.

Page 7:	<code>\index{gnat!size of}</code>	gnat, 32
Page 32:	<code>\index{gnat}</code>	anatomy, 35
Page 35:	<code>\index{gnat!anatomy}</code> <code>\index{gnus!good}</code>	size of, 7 gnus
Page 38:	<code>\index{gnus!bad}</code>	bad, 38 good, 35

You can also have subsubentries.

Page 8:	<code>\index{bites!animal!gnats}</code>	bites
Page 10:	<code>\index{bites!animal!gnus}</code>	animal
Page 12 :	<code>\index{bites!vegetable}</code>	gnats, 8 gnus, 10 vegetable, 12

$\LaTeX$  and *MakeIndex* support only three levels of indexing; you can’t have subsubsubentries.

To specify a page range, put an `\index{...|{}` command at the beginning of the range and an `\index{...|})}` command at the end of the range.

Page vi:	<code>\index{gnat {}</code>	gnat, vi–x, 22
Page x:	<code>\index{gnat })}</code>	gnus
Page 22:	<code>\index{gnat}</code> <code>\index{gnus!bad {}</code> <code>\index{gnus!bad })}</code>	bad, 22 good, 28–32
Page 28:	<code>\index{gnus!good {}</code>	
Page 30:	<code>\index{gnus!good}</code>	
Page 32:	<code>\index{gnus!good })}</code>	

---

<sup>2</sup>To avoid any ambiguity if a page break occurs right at an `\index` command, it’s a good idea to attach the command to a word instead of putting it between words.

Note that *MakeIndex* does the right thing when both ends of the range fall on the same page.<sup>3</sup>

Sometimes, you want to add a cross-reference with no page number. This is done as follows;

```
Page 2: \index{at}           |||      at, 2
Page 2: \index{at!bat|see{bat, at}} |||      bat, see bat, at
```

Since the “see” entry does not print any page number, it doesn’t matter where the `\index{...|see{...}}` command goes, so long as it follows the `\begin{document}` command. You might want to put all such cross-referencing commands in one place.

If you specify an entry of the form  $\sigma@tau$ , the string  $\sigma$  determines the alphabetical position of the entry, while the string  $\tau$  produces the text of the entry.

```
Page 44: \index{twenty}      |||      twenty, 44
Page 46: \index{twenty-one}  |||      xx, 55
Page 55: \index{twenty@xx}   |||      twenty-one, 46
```

This feature is useful because the argument of the `\index` command provides the actual input string that L<sup>A</sup>T<sub>E</sub>X uses to generate the index entry. In the following example, the input `\alpha` produces the symbol  $\alpha$ ,

```
Page 12: \index{alphas}      |||      alpha, 13
Page 13: \index{alpha}       |||       $\alpha$ , 14
Page 14: \index{alpha@$\alpha$} |||      alphas, 12
```

Similarly, the command `\index{gnu@{\bf gnu}}` produces a boldface **gnu** index entry.

In some indexes, certain page numbers are specially formatted—for example, an italic page number may indicate the primary reference, and an *n* after a page number may denote that the item appears in a footnote on that page. *MakeIndex* makes it easy to format an individual page number any way you want. For any string of characters  $\sigma$ , the command `\index{...|\sigma}` produces a page number of the form  $\sigma\{n\}$ . Similarly, the command `\index{...|(\sigma)}` may produce a page number of the form  $\sigma\{n-m\}$ . Thus, suppose the document contains the following command definitions:

```
\newcommand{\ii}[1]{\it #1}
\newcommand{\nn}[1]{#1n}
```

We could then have:

```
Page 3: \index{gnat|ii}      |||      gnat, 3, 4n
Page 4: \index{gnat|nn}      |||      gnu, 5, 44-46
Page 5: \index{gnu}          |||
Page 44: \index{gnu|(ii)}    |||
Page 46: \index{gnu|)}       |||
```

The “see” option is a special case of this facility, where the `\see` command is predefined by the `makeidx` document-style option.

---

<sup>3</sup>The use of “28ff.” rather than “28–32” is frowned upon by the experts.

**A.4.2.3 The Fine Print** Commands in an index entry are expanded when the index is typeset, not when the `idx` file is written. Hence, the command `\index{\gnu}` produces an entry that is alphabetized by `\gnu`, regardless of how the `\gnu` command is defined.

Recall that special characters like `\` may appear in the argument of an `\index` command only if that command is not itself contained in the argument of another command. This is most likely to be a problem when indexing items in a footnote. Even in this case, robust commands can be placed in the “@” part of an entry, as in `\index{gnu@\it gnu}`, and fragile commands can be used if protected with the `\protect` command.<sup>4</sup>

Remember that the argument of an `\index` command must always have matching braces, where the brace in a `\{` or `\}` command counts.

*MakeIndex* assumes that all page numbers are either arabic or lower-case roman numerals; it assumes that pages numbered with roman numerals precede those numbered with arabic numerals.

To put a `!`, `@`, or `|` character in an index entry, *quote* it by preceding the character with a `"`. More precisely, any a character is said to be quoted if it follows an unquoted `"` that is not part of a `\` command. A quoted `!`, `@`, or `|` character is treated like an ordinary character rather than having its usual meaning. The `"` preceding a quoted character is deleted before the entries are alphabetized.

Page 2:	<code>\index{exclaim ("!)}</code>		exclaim (!), 2
Page 3:	<code>\index{exclaim ("!")!loudly}</code>		loudly, 3
Page 4:	<code>\index{fur@f\{"u}r}</code>		für, 4
Page 5:	<code>\index{quote (\verb+""+)}</code>		quote (""), 5

*MakeIndex* regards spaces as ordinary characters when alphabetizing the entries and deciding whether two entries are the same. Thus, letting “`␣`” denote a space character, the commands `\index{gnu}`, `\index{␣gnu}`, and `\index{gnu␣}` produce three separate entries, the first appearing near the beginning of the index, since `␣` comes before any letter in *MakeIndex*’s “alphabetical” order. All three entries look the same when printed, since  $\LaTeX$  ignores extra spaces in the input. Similarly, `\index{a␣space}` and `\index{a␣␣space}` produce two different entries that look the same when printed. Do not split the argument of an `\index` command across lines in the input file.

### A.4.3 Error Messages

*MakeIndex* types out on the terminal the number of lines read and written and how many errors were found. Message to identify the error are written on a file with extension `ilg`. There are two phases in which *MakeIndex* can produce error messages: when it is reading the `idx` file, and when it is writing the `ind` file. Each error message prints the nature of the error followed by a line number, identifying where in the file the error occurs. In the reading phase, the line number refers to the `idx` file; in the writing phase, it refers to the `ind` file.

---

<sup>4</sup>In versions of  $\LaTeX$  released before 3 February 1987, you must use the `\string` command instead of `\protect`.

### A.4.3.1 Errors in Reading Phase

**Extra ‘!’ at position ...** The `\index` command’s argument has more than two unquoted `!` characters. Perhaps some of them should be quoted.

**Extra ‘@’ at position ...** The `\index` command argument has two or more unquoted `@` characters with no intervening `!`. Perhaps one of the `@` characters should be quoted.

**Extra ‘|’ at position ...** The `\index` command’s argument has more than one unquoted `|` characters. Perhaps the extras should be quoted.

**Illegal null field** The `\index` command argument doesn’t make sense because some string is null that shouldn’t be. The command `\index{!big}` will produce this error, since it specifies a subentry “big” with no entry. Similarly, the command `\index{@big}` is incorrect because it specifies a null string for alphabetizing.

**Argument ... too long (max 1024).** Your document contained an `\index` command with a very long argument. You probably forgot the right brace that was supposed to delimit the argument.

**Other errors** *MakeIndex* can produce a variety of other error messages indicating that something is seriously wrong with the `idx` file. If you get one, it probably means that the `idx` file was corrupted in some way. If  $\LaTeX$  did not generate any errors when it created the `idx` file, then it is highly unlikely to have produced a bad `idx` file. If it did, you’ll have to examine the `idx` file to figure out what went wrong.

### A.4.3.2 Errors in Writing Phase

**Unmatched range opening operator** An `\index{...|}` command was not followed by a matching `\index{...|}` command. The “...” in the two commands must be completely identical.

**Unmatched range closing operator** An `\index{...|}` command was not preceded by a matching `\index{...|}` command.

**Extra range opening operator** Two `\index{...|}` commands appear in the document with no intervening `\index{...|}` command.

size	default (10pt)	11pt option	12pt option
<code>\tiny</code>	5pt	6pt	6pt
<code>\scriptsize</code>	7pt	8pt	8pt
<code>\footnotesize</code>	8pt	9pt	10pt
<code>\small</code>	9pt	10pt	11pt
<code>\normalsize</code>	10pt	11pt	12pt
<code>\large</code>	12pt	12pt	14pt
<code>\Large</code>	14pt	14pt	17pt
<code>\LARGE</code>	17pt	17pt	20pt
<code>\huge</code>	20pt	20pt	25pt
<code>\Huge</code>	25pt	25pt	25pt

Table 3: Type sizes for L<sup>A</sup>T<sub>E</sub>X size-changing commands

**Inconsistent page encapsulator ... within range** *MakeIndex* has been instructed to include a page range for an entry and a single page number within that range that is formatted differently—for example, by having a `\index{gnu|ii}` command between a `\index{gnu|}` and a `\index{gnu|})` command.

**Conflicting entries** *MakeIndex* thinks it has been instructed to print the same page number twice in two different ways—for example, by the commands `\index{gnu}` and `\index{gnu|(see{...})}` appearing on the same page.

## Acknowledgements

*MakeIndex* is a C program written by Pehong Chen, with a little advice from me. It was inspired by a program written by Mike Urban, which was based on a program written by Marshall Rose. Perhaps someday someone will rewrite *MakeIndex* in Web, making it available to all L<sup>A</sup>T<sub>E</sub>X users.

## A.5 Fonts

Almost all the symbols available on our fonts can be generated by ordinary L<sup>A</sup>T<sub>E</sub>X commands. However, there are type sizes not obtainable by L<sup>A</sup>T<sub>E</sub>X's size-changing commands with the ordinary document styles. Consult a local T<sub>E</sub>X expert to find the T<sub>E</sub>X name for such a font.

Tables 3 and 4 allow you to determine if the font for a type style at a particular size is preloaded, loaded on demand, or unavailable.

Table 3 tells you what size of type is used for each L<sup>A</sup>T<sub>E</sub>X type-size command in the various document-style options. For example, with the 12pt option, the `\large` declaration causes L<sup>A</sup>T<sub>E</sub>X to use 14pt type. Table 4 tells, for every type size, to which class of fonts each type style belongs. For example, in 14pt type, `\bf` uses a preloaded font and the other five type-style commands use load-on-demand fonts. Roman (`\rm`) and math italic (`\mit`) fonts are all preloaded; the `\em` declaration uses either italic (`\it`) or roman.

	' it	' bf	' sl	' sf	' sc	' tt
5pt	D	D	X	X	X	X
6pt	X	D	X	X	X	X
7pt	P	D	X	X	X	X
8pt	P	D	D	D	D	D
9pt	P	P	D	D	D	P
10pt	P	P	P	P	D	P
11pt	P	P	P	P	D	P
12pt	P	P	P	P	D	P
14pt	D	P	D	D	D	D
17pt	D	P	D	D	D	D
20pt	D	D	D	D	D	D
25pt	X	D	X	X	X	X

Table 4: Font classes: P = preloaded, D = loaded on demand, X = unavailable.

## A.6 Summary of Available Programs

What follows is a selection of interesting programs currently installed in `/usr/bin` or `/usr/grads/bin`.

`amstex` T<sub>E</sub>X with AMS styles preloaded

`bibtex` Bibliography processor for L<sup>A</sup>T<sub>E</sub>X see L<sup>A</sup>T<sub>E</sub>X book

`detex` Strip T<sub>E</sub>X control sequences, usually before checking spelling

`dvips` DVI to PostScript filter

`fig2dev` Convert Fig files to L<sup>A</sup>T<sub>E</sub>X, Postscript, *etc.*

`ispell` Interactive spelling corrector, has a mode for T<sub>E</sub>X

`latex` See the L<sup>A</sup>T<sub>E</sub>X book

`makeindex` Index processor for T<sub>E</sub>X, L<sup>A</sup>T<sub>E</sub>X and other languages

`mf` Plain METAFONT

`mpost` Plain METAPOST

`psselect` Select subset of pages in Postscript file

`tex` See the T<sub>E</sub>X book

`texexpand` Expands included files in T<sub>E</sub>X input

`texmatch` Checks matching in T<sub>E</sub>X documents

`xdvi` DVI previewer for X Windows

`xfig` Graphical editor under X11

## A.7 Summary of L<sup>A</sup>T<sub>E</sub>X Macros

Local styles found in `/usr/local/share/texmf/tex/latex/urcs`: for more information see the “Guide to Department Styles,” Department Guide 13.

`urcsaddendum.tex` Errors from the L<sup>A</sup>T<sub>E</sub>X book, 1988

`urcsannotated.bst` BIB<sub>T</sub><sub>E</sub>X style that allows an `annote` key

`urcsbiblianno.bst` BIB<sub>T</sub><sub>E</sub>X style that combines `urcsbiblio` and `urcsannotated`

`urcsbiblio.bst` BIB<sub>T</sub><sub>E</sub>X style for citations of the form “(Name, Date)” and non-reversed names in references

`urcsbiblio.sty` Style files required for use with `urcsbiblio` format bibliographies

`urcscover.sty` Formats the cover of a department technical report

`urcscoverrs.sty` Formats the cover of a student research summary

`urcsdraft.sty` Style that provides a header indicating when the file was last modified

`urcslogo.tex` The department logo, as used on TR’s, using Postscript fonts

`urcsmacros.tex` A variety of macros, included by most URCS styles

`urcsmanual.sty` Style for a large manual

`urcsmargin.sty` Margin and spacing settings for department styles

`urcsnoversions.sty` Turn off all versioning provided by `urcsversions`

`urcsreport.sty` Style for department technical reports

`urcssectindent.sty` Something of Lawrence’s

`urcsstyles.tex` Source to the “Guide to Department Styles”

`urcssummary.sty` Style for student research summaries

`urcsthesis.sty` Style for department dissertations, apparently ok with the dean, at least the last time we checked

`urcstitlepage.sty` Makes the titlepage for department styles

`urcsversions.sty` Provide version indicators in side margin

## A.8 Common Errors

This is a local section. Here are some common errors made by users. Profit from those who have slipped before you. These are not bugs because they are documented behavior.

Send in your favorite gotchas for the next version of this manual.

### A.8.1 Header Commands

**The 10pt Option** — There is no 10pt option on `\documentstyle`. It is the default. Yes, this is non-uniform, but ...

**Roman Numbering for Table of Contents** — There are a number of approaches to printing the table of contents on pages with roman numerals. First, use a style that provides this, such as `urcsmanual`. Second, use the following commands:

```
...
\pagenumbering{roman}
\maketitle
...
\tableofcontents
...
\chapter{First and Only First}
\pagenumbering{arabic}
...
```

Third, place the following commands at the end of the document:

```
...
\newpage
\pagenumbering{roman}
\tableofcontents
\end{document}
```

### A.8.2 Sections

**First Paragraph Indentation** — The first paragraph of a section is not indented. This is the preferred typographical style. If you must indent, use the `urcssectindent` style option.

**Footnotes** — Footnotemarks in section titles need a `protect`. For example,

```
\section{Introduction\protect\footnotemark}
\footnotetext{This bulk of this chapter first appeared as
\cite{fanty85}.}
```

**Automatic Numbering** — To avoid automatic section numbering, use the `\section*` command instead of the `\section` command. There are star forms for the other subdivisions too.

**Numbering Depth** — You may control the depth of section numbering with the command

```
\setcounter{secnumdepth}{2}
```

where 2 corresponds to numbering subsections but no lower.

### A.8.3 Spacing

**Double Spacing** — To get double spacing, use the `doublespace` style option.

**Multiple Newlines** — One or more spaces act as a single space. One newline acts as a single space. Two or more newlines acts as the `\par` command to start a paragraph.

**Space at Top of Page** — The commands `\vskip 3in` and `\vspace{3in}` do not generate space at the top of the page, use the command `\vspace*{3in}` instead.

**Blank Pages** — The commands `\vfill\eject` do not generate a blank page; `~\vfill\eject` do.

**Multiple Parameter Commands** — A newline is a space, and a single space can be a parameter, this distinction matters a great deal when a command takes more than one long parameter. For example,  $\TeX$  interprets

```
\newenvironment{foo}
{blah}
{barf}
```

as passing the parameters `{foo}`, `space`, and `{blah}` to the `\newenvironment` command. The appropriate form is:

```
\newenvironment{foo}%
{blah}%
{barf}
```

In case you are wondering, the `%` is the comment character and swallows the newline.

### A.8.4 Figures and Tables

**Long Captions** — The limit on the length of captions is the size of the input buffer, typically 500 characters. This is documented and due to the way  $\LaTeX$  does captions.

**Labels** — Labels on tables and figures must follow the corresponding caption.

**Nested Tabbing** — Tabbing environments don't nest. However, you can use the `\pushtabs` and `\poptabs` commands.

**Boxed Text** — It is sometimes difficult to get a boxed figure of formatted text inserted. Here is one solution:

```
\begin{figure}
\begin{quotation}
\framebox{
\parbox{4 in}{
INSTRUCTIONS: Circle the number corresponding to the value
```

associated with your evaluation of the manager on the dimension defined below.

\\

\\

LONG-RANGE PLANNING: Forecasts with respect to manpower planning and costs; anticipates future problems and new trends; reviews with his people to constantly update them and uncover new information.

\\

\\

```
\begin{tabular}{ccccc}
```

```
1 & 2 & 3 & 4 & 5 \\
```

```
Unsatisfactory & Fair & Good & Very Good & Exceptional \\
```

```
\end{tabular}
```

```
} % \parbox
```

```
} % \framebox
```

```
\caption{Example of a graphic rating scale  
with numerical and verbal anchors.
```

```
(Source: Berk, Ronald A., ed. \emph{Performance Assessment},  
The Johns Hopkins University Press, 1986.)}
```

```
\end{quotation}
```

```
\end{figure}
```

## A.9 Bugs

There are a few known bugs in  $\text{\LaTeX}$  that occur very seldom and cause the user little trouble, but would be very difficult to fix. Moreover, given the nature of complex systems, it is not unlikely that the corrections would lead to even worse problems. Therefore, these bugs will probably not be fixed.

The bugs and ways to get around them are listed below. Do not worry about any of them until you are preparing the final draft, since changes to the text are very likely to cause the problem to disappear.

- In rare instances, a figure or table will be printed on the page preceding the text where the `figure` or `table` environment appears. This can be fixed by moving the environment further towards the end of the document.
- A marginal note at the top of a page may appear in the wrong margin. This can be fixed by inserting a redundant `\pagebreak` command to force a page break exactly where  $\text{\LaTeX}$  started the new page anyway.
- A footnote can be broken across two pages when it should fit on a single page. This happens when there is one or more figures or tables on the page. The problem is corrected by moving, towards the end of the file, the last `figure` or `table` environment that produces a figure or table on the page where the footnote starts.