Comments on using LaTeX for theses

Federico Garcia*

August 5, 2003

Contents

*For Pitt's ETD Working Group

Introduction

This document introduces some LaTeX tools that may or not be known to all LaTeX users in the Pitt community, and that might prove useful to prepare their electronic theses or dissertations (ETD). Thus, it is complementary to the documentation of the newly-created 'document class' pittetd, although its contents is in no way limited to this class (nor to theses and dissertations, for that matter).

Motivation to write these *Comments* arose from questions I received from users that answered the ETD Working Group's survey, and especially from those who offered themselves as volunteer testers for pittetd. I suspect that most LaTeX users will find below many 'old news' of little or no interest; but I am also confident most of them will get to know of new possibilities previously unknown to them. On the other hand, since I am myself nothing else than one more LaTeX user, this document will inevitably be confined to what is already old news for me—surely many things will be omitted simply because I have not heard of them.

Main parts of this document are devoted to inclusion of graphics, that presents serious problems to ETD's because of the unfortunate and unbridgeable incompatibility that exists between the PostScript and the PDF formats; to the automatic compilation of indices with the program *MakeIndex*; and to the tools that LaTeX offers for the handling of large documents. Less-than-deserved attention is paid to BibTeX, because it is well documented by its author [4], and because

the immense variety of styles makes it impossible to even dream of a useful survey.

The contents is arranged in paragraphs (§1, §2, etc.), rather than sections, to allow a quick reference from the table of contents. Accordingly, the final index gives paragraph numbers, not page numbers. The paragraphs are grouped, though, under section-like headings.

This is not an 'advanced' guide to LaTeX, but neither is it an 'introduction.' The reader is assumed to know of and be acquainted with the use of most if not all of LaTeX's commands and environments. Information on all that can be found in the file `sample2e.tex`, distributed with LaTeX, and of course in the LaTeX user's manual [2], owned by the Pitt library, but always checked out (and not by me). From the pittetd webpage at `http://www.pitt.edu/~graduate/etd/latextemplate.html`, other guides can be downloaded.

There is a glossary toward the end of this document. And, before starting, a final note: the character ␣ stands for a blank space where emphasis is needed.

General Edition

Punctuation and spacing

No insistence on the next points of text edition in LaTeX will ever be too much. LaTeX is capable of much finer tuning than word processors, but this in turn requires more control from the user. The correct use of quotation marks (i.e., the use of the key `‘`, seldom meaningful in other programs) is obvious enough to require no extra warning, but there are other subtleties that should be addressed.

§1. *Dashes* are one of them. *Em*-dash—the punctuation mark—is produced by three dashes `---`. *En*-dashes, instead, are used for ranges, as in 'W. Lutosławsky, 1913–1994.' And the regular dash is used for compound words: 'twentieth-century capitalism.' A common mistake is to use en-dash as word-separator, in ugly constructions such as 'word–separator.'

§2. *Inter-word spacing* is automatically handled by LaTeX; among other things, it assigns more space after a period than after most other characters. Sometimes, however, when a period is not a sentence marker, no extra space is wanted. It is generally well known that this
\ is avoided by using the 'space command,' `\␣`:

```
This␣is␣Dr.␣Freud.
```
This is Dr. Freud.

```
This␣is␣Dr.\␣Freud.
```
This is Dr. Freud.

What is much less well known is that LaTeX does not apply extra space after uppercase letters, for example in 'Donald E. Knuth.' This is because, in general, a sentence does not end with an uppercase letter, so LaTeX assumes the period is not a sentence marker. So, against what many people think, the result of `M.\␣A.` is exactly the same as that of `M.␣A.`

But, what if a period-after-uppercase is in fact a final period? How
\@ is the extra space forced? The obscure command `\@` is intended just for that. Here is an illustration:

```
...see appendix C. For the moment, ...
```
...see appendix C. For the moment, ...

```
...see appendix C\@. For the moment, ...
```
...see appendix C. For the moment, ...

§3. *Ties* (the '~' symbol) are a related concept. A tie produces a space of regular width (exactly as `\ `), but in addition it 'ties' the two words together, so that the line will not be broken by LaTeX at the space between them. This is imperative in abbreviations at the very least, for it would be absurd to allow the expression 'M. A.' to be broken as in 'M. A.' (this tends to happen in Word all the time...). Typing `M.~A.` avoids it. Personally, I use ties in many more contexts, such as names, multiple-word concepts, titles, etc., although many well-edited books can be found where this things are separated.

§4. *Acronyms* are all-uppercase expressions such as 'ETD,' and also call for special treatment. It is a feature of TeX's Computer Modern fonts (as of most fonts) that uppercase letters are sensibly larger than lowercase, so that NASA does not look well. In fact, NASA (typeset in a smaller font) is much better. In principle, this seems easy to achieve just by changing the font size: `{\small NASA}`. But this

is not a robust solution: in a footnote (whose text is smaller than `\small`), the result is contrary to the need, and when the text moves from one part of the document to another (for example, in section titles that go to the Table of Contents), the font size might change
`\acro` and ruin everything. So there exists the `\acro` command that finds the current font size and selects a smaller one for the expression: you type `\acro{NASA}`. This command is *not* part of the LaTeX kernel, and therefore it must be manually provided. The pittetd class has it built in, so if you are typing your thesis with it, the command is available (see Appendix A for what you have to type to use this command in standard classes).

§5. *Ellipsis*[1] is normally produced by the command `\dots`... However, this command is not very reliable as regards the spacing before and after the three dots, or their effect on surrounding punctuation marks. Matt Swift has created a small package that helps to format
`\lips` text ellipses: the `lips` package, that provides the `\lips` command, a substitute for `\dots`. As an illustration, I copy Figure 2 from the package's documentation as Appendix B below.

Definition of commands

§6. The `\newcommand` command allows the user to create his/her own commands. One particular kind of command with no arguments is what can be called '*abbreviation commands*,' that facilitate the typesetting of frequently-occurring expressions:

```
\newcommand{\pcs}{pitch-class set}
\newcommand{\stm}{Static Timing Analysis}
\newcommand{\cubii}{\emph{Clavier-\"Ubung~II}}
```

While these abbreviations are really helpful (they also ensure similar formatting for similar expressions), every LaTeX user has had to cope with the annoying fact that they gobble the space that follows: '`\pcs number 4`' produces the wrong output 'pitch-classnumber 4'.

[1] Note that this paragraph apply to 'text' ellipses, not to mathematical expressions.

Of course, "embedding" the space into the definition (something like `\newcommand{\pcs}{pitch-class␣set␣}`) does not work, for '`this \pcs, whose`' generates 'this pitch-class set , whose'...

§7. ***xspace*** is the small but brilliant package that handles this. It is
`\xspace` typically distributed with LaTeX, and provides the `\xspace` command, so that you can type rather

```
\newcommand{\pcs}{pitch-class set\xspace}
\newcommand{\stm}{Static Timing Analysis\xspace}
\newcommand{\cubii}{\emph{Clavier-\"Ubung~II}\xspace}
```

and the abbreviations will produce a space only if they are not followed by a punctuation mark. That way you can issue both `\pcs␣` and `\pcs,␣` and get the right spacing. (To cancel a space when `\xspace` would produce it, you can enclose the next character within brackets: `\pcs{$_2$}`.)

§8. The other minor problem when defining abbreviation commands is *math mode*. If you define

```
\newcommand{\#}{$\sharp$}
```

you will be able to use the command `\#` (to produce '♯', only possible in math mode) in regular text... but if you use it in math mode (like `$a_\#$`), LaTeX will complain. Conversely, if you do not include the `$`'s in the definition, the command could not be used in regular text.

`\ensuremath` LaTeX offers a way around this: the `\ensuremath` command. You should define

```
\newcommand{\#}{\ensuremath{\sharp}}
```

(where `\ensuremath{...}` replaces `$...$`) and the command will behave properly in both text and math.

§9. *Example*: if you write a lot about ℕ, you can make use of both `\xspace` (§7) and `\ensuremath` (§8) and achieve a really robust command `\N` by way of

```
\newcommand{\N}{\ensuremath{\mathbb{N}}\xspace}
```

Hidden possibilities

LaTeX has a couple things that—one reason or another—tend to pass unnoticed until one discovers them serendipitously.

\\ §10. *The optional argument to* \\ is one of them. In virtually every context where this line-breaking command is used (regular text, `array` or `tabular` environments, etc.), you can add a LaTeX dimension (within square brackets, as in `\\[3mm]`), which will be vertically added to the line skip. For example:

```
Chess\\ Checkers\\ Canasta\\ Bridge
```

Chess
Checkers
Canasta
Bridge

```
Chess\\ Checkers\\[2mm] Canasta\\ Bridge
```

Chess
Checkers

Canasta
Bridge

The dimension can also be negative.

§11. *Stacking words* (or any kind of LaTeX box) on top of each other is sometimes needed, for things like  . It would seem that the only way to do this is by means of `\parbox`:

```
\fbox{\parbox{3mm}{\scriptsize\textsf{A}$\clubsuit$}}
```

but this of course implies 'guessing' the right width for the box (in this case there is no problem, but when you deal with words or formulas, you are bound to do it by trial and error). Another possibility is to try with a one-column `tabular` environment (which automatically finds the width of the columns), but that involves a lot of typing and uncertainty about spurious vertical spaces. Well, the `\shortstack` command will do nicely:

\shortstack

```
\fbox{\scriptsize\shortstack{\textsf{A}\\$\clubsuit$}}
```

Different lines are separated by \\ (whose optional argument functions here as well), and LaTeX finds the width of the box. By default,

the lines will be horizontally centered, but left- or right-justification can be forced with an optional argument: `\shortstack[l]{...}` or `\shortstack[r]{...}`.[2]

The only drawback to `\shortstack` is that, respect to the surrounding text, the box it creates will always be aligned by its base, as in some stacked words . To achieve some stacked words, I had to use `\raisebox`.

§12. *Dimensions* are often the argument of important commands, such as `\parbox` or `p{...}` in the format part of a `tabular` environment. While the most natural way to give LaTeX the dimension it asks for is absolute units (cm, in, etc.), it is important to realize that LaTeX 'lengths' can be used instead. For example, `\textwidth` holds the width of a regular line of text, and it can be very useful when designing a table, say of 3 columns. Rather than guessing their width in centimeters, you can simply distribute the total width between the margins by specifying `.2\textwidth`, `.4\textwidth`, and `.4\textwidth`, for example. A list of such useful LaTeX lengths is:

`\textwidth`	The width of a regular line of text
`\textheight`	The height of the text on a regular page
`\paperwidth`	The width of the page (including margins)
`\paperheight`	The height of the page
`\parindent`	The width of the paragraph indentation

§13. *Centering and flushing* text to the left or to the right can be done by the three environments `centering`, `flushleft` and `flushright`; however, all of the three add a vertical space below and after their contents, which sometimes is not welcome (for example, inside a `\parbox`, or in a table). For those situations, there exist the declarations `\centering`, `\raggedleft`, and `\raggedleft`.

`\centering`
`\raggedleft`
`\raggedright`

Compare the space before and after the present paragraph (produced by `{\centering Compare the space...}`)

[2] The `\stackrel` command works similarly, but it is for math mode, and the lines are always horizontally centered.

and that before and after the one below.
The next paragraph is typeset inside a `center` environment:

And therefore it is separated by the preceding and following text
even if there is no blank lines in the input file.

§14. *Text in math mode* is usually produced by the `\text` command of the `amsmath` package. But this is actually a redundant feature, and the package does not have to be loaded to make it easy to typeset text within mathematical formulas. All that is needed is the LaTeX command `\mbox`:

```
\[0<a_n<\frac{1}{n}\quad
\mbox{for every natural } n\ge 1.\]
```

has the effect

$$0 < a_n < \frac{1}{n} \quad \text{for every natural } n \ge 1.$$

On cross-referencing

Cross-referencing with the commands `\label{`⟨*key*⟩`}` and `\ref{`⟨*key*⟩`}` is straightforward. The latter will print the counter of the element to which a corresponding `\label` was assigned. Among the elements are document divisions (chapters, sections, etc.), floating objects (tables and figures), and equations.

§15. *Referring to footnotes* is less well-known. But the fact is that `\label` can also be assigned to a footnote (by typing it *after* the footnote), and then a coupling `\ref` will print the footnote number. For example, since I added `\label{fn}` right after the last footnote of this document, I can now say that it is footnote number 8:

```
...footnote number~\ref{fn}.
```

§16. *Referring to items* of an `enumerate` environment is also possible. For example, after

```
\begin{enumerate}
\item\label{en:1} First item.
```

```
    \begin{enumerate}
    \item\label{en:21} Item in the second level.
    \end{enumerate}
\item Second item.
\end{enumerate}
```

all is set to refer to the `\label`'ed items with `\ref`.

`\pageref` §17. *Page number referencing* is also available, through the `\pageref` command. It will typeset the number of the page in which the corresponding `\label` occurred, rather than the counter of the element. Since `\pageref` does not depend on counters of any kind (other than the page number, of course), it can be used to refer to *any* kind of text. In other words, a `\label` command can be put anywhere in the document, even within running text, and a `\pageref` command will work well. (By contrast, `\ref` will produce the last referrable counter—divisions, equations, floating objects, enumerate items, or footnotes—that occurred before the `\label` command.)

§18. *Name referencing* is an option provided by the `hyperref` package (through its subordinate `nameref` package), and it makes sense when
`\nameref` references are interactive links. The `\nameref` command, to be used instead of `\ref`, typesets the *name*, not the number, of the chapter or section referred to. This command only work with divisions.

Since the titles of chapters and sections should be all-uppercase for Pitt ETD's, it is a good idea to use `\acro` (§4) when referring to them with `\nameref`. Thus, `\acro{\nameref{intro}}` would produce the better 'INTRODUCTION', not 'INTRODUCTION'.

§19. *Making keys visible*, so that they do not have to be remembered and can be easily identified, is very useful when writing documents with many `\label{`⟨*key*⟩`}`-`\ref{`⟨*key*⟩`}` (or similar) constructions. The `showkeys` package is designed for that purpose; when loaded, it will make keys defined with `\label` appear in the margins, while keys referred to by `\ref` and related commands appear as superscripts. In neither case the actual layout of the text is affected. The keys of the `\cite` commands are also visible.

When the `final` option is used, either as an option to the package or one to the class—to `pittetd`, for example—the keys will not be

visible. By this mechanism the user is spared from having to comment out the `\usepackage{showkeys}` line.

Large documents

§20. A good practice when writing large documents is to divide them up into several files. It saves processing time, and provides for quicker access and better organization. The document is thus actually a whole 'project,' consisting of a *main file* and several *subsidiary files* that are loaded by the former.

Three ways of handling such projects in LaTeX are presented in the following paragraphs. Each has its own scope and is applicable to its own kind of situations.

§21. ***`\include`'ing files*** is the first way, done with the LaTeX command `\include`. In principle,

`\include`

```
\include{⟨filename⟩}
```

calls the subsidiary file called ⟨*filename*⟩. The extension `.tex` is assumed (and in fact should not be included in the ⟨*filename*⟩). There can be naturally many `\include` commands in a document, so that different chapters can be typeset as different files. The main file, if this mechanism is used systematically, could then consists of little more than the preamble and a series of `\include` commands, something like:

```
\documentclass[dvipdfm]{pittetd}
```

⋮ } ⟸ *preamble*

```
\begin{document}
\include{prelims}
\include{intro}
\include{cap1}
\include{cap2}
\include{cap3}
```

```
\appendix
\include{appA}
\include{appB}
\include{biblio}
\end{document}
```

§22. *Excluding files*. The most 'elegant' thing about this procedure is
`\includeonly` the 'exclusion' mechanism `\includeonly`. This command—that has to be issued within the preamble—limits the list of files to be included, thus saving processing time when isolated modifications are made to individual files. For example, by adding to the main file the line

```
\includeonly{cap1,appA}
```

LaTeX will only process the files `cap1.tex` and `appA.tex`, *keeping* however the numbering of pages, chapters, figures, tables, etc., and all cross references and citations, as if the rest of the files were present. Needless to say, after an `\includeonly` command there is no necessity of deleting or commenting out the `\include` commands.

§23. *Warnings about* `\include` have to be made: the command always starts a new page. This makes sense for chapters, but is of little help for figures, parts of chapters, etc. Also, `\include` cannot be present in `\include`'d files ('nesting' is not allowed).

§24. *`\input`'ting files*. When a document has figures or other constructions of certain complexity, whose LaTeX code takes lines and lines of unintelligible commands, it is usually a good idea to separate them from the text, putting them in subsidiary files. This adds to the clarity of the text and to the accessibility of the figures or complex constructions.

The `\include` mechanism, however, would not work well to handle this kind of relationship between main and subsidiary files, because
`\input` each subsidiary file starts in a new page. But the command `\input` is a good alternative. A line like

```
\input{fig1.tex}
```

invokes the file `fig1.tex`, and reads it exactly *as if it were part of the main file*. It will not start a new page, not even a new line. But then again, removing the line is completely equivalent to having deleted the figure itself—cross referencing, page numbering and

layout change accordingly. There is no 'exclusion mechanism' similar to `\includeonly`. On the other hand, `\input` can be nested (it can appear in files that are themselves `\include`'d or `\input`).

§25. *Working on individual files*. Another advantage of separating complex constructions from the main file is that you can work on them—fine-tune them to perfection—without processing the whole project. In other words, you can create the figure as a stand-alone document, and only `\input` it into the main file when it's ready. Before making this last step, however, you have to delete (or comment out) the preamble of the subsidiary file, as well as its `\end{document}`. But then again, if you want to come back and edit the figure, you will have to re-write (or un-comment) those things...

§26. *The* `subfiles` *package* was created[3] to facilitate this process. By using it, subsidiary files can be processed *either* on their own, *or* as part of the main file, with no modification needed.

To put `subfiles` at work, the 'subfile(s)' should begin with the lines

```
\documentclass[⟨main_file⟩]{subfiles}
\begin{document}
```

and takes the preamble from the ⟨*main_file*⟩ (loading the same packages and setting the same parameters as the main file). They can thus be processed as a regular LaTeX files.

`\subfile` `\subfile` Then, in order to invoke it into the main file (which has loaded the package by means of `\usepackage{subfiles}`), the `\subfile` command is used:

```
\subfile{fig1.tex}
```

and no change to `fig1.tex` is needed, since the main document will ignore the `\documentclass`, `\begin{document}` and `\end{document}` commands of the subsidiary file.

The `\subfile` command resembles `\input` more closely than `\include`: it does not start a new page, it can be nested, and there is no exclusion mechanism like `\includeonly`.

[3] By me, actually, in 2002.

Inclusion of graphics

§27. *The* ***graphicx*** *package*, an extension and improvement on the previous `graphics` package, and present in typical distributions of LaTeX, offers tools for—among other things—inclusion of imported graphics of a variety of formats. The package has to be loaded with an option corresponding to the graphic 'driver' to be used. For regular DVI documents, the most common driver is `dvips`. Some implementations of LaTeX come with their own driver (Macintosh's *Textures* and *OzTeX*, for example, offer drivers `textures` and `oztex`, respectively).

For example, to load the `graphicx` with `dvips` driver, you type (in the preamble):

```
\usepackage[dvips]{graphicx}
```

Relevant for ETD's (that are PDF files) are the drivers `dvipdfm` and `oztex` and `pdftex`. When one of these two options has been specified for pittetd (for example with `\documentclass[dvipdfm]{pittetd}`), the latter will pass it along to all the packages, so it is enough to write `\usepackage{graphicx}`.[4] This makes it possible to change the driver of pittetd without having to change that of the packages.[5]

§28. *Including graphics* is pretty straightforward with the `graphicx` package through its `\includegraphics` command. Here is the syntax:

`\includegraphics`

```
\includegraphics[⟨options⟩]{⟨filename⟩}
```

A list of ⟨*options*⟩ is given in Table 1 (an adaptation of [1, Table 7.3]).

The ⟨*filename*⟩ should in principle include the whole path of the file, unless a generic declaration has been made with `\graphicspath`. For example, in Windows, the command

`\graphicspath`

```
\graphicspath{{c:/thesis/graphics}{c:/old papers}}
```

allows writing only the name in the ⟨*filename*⟩ of `\includegraphics`. LaTeX will search the given directories for the required file.

The following paragraphs discuss some of the ⟨*options*⟩.

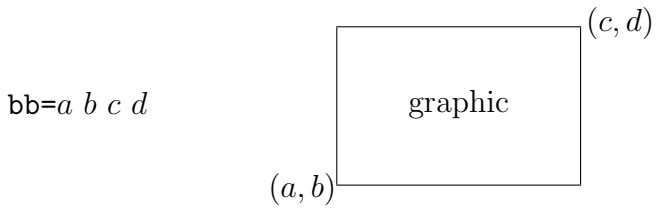
[4] There are more packages that need to know what the driver is, notably `hyperref` and `color`.

[5] The files `dvipdfm.def` and `pdftex.def` should be available to LaTeX, preferably in the same directory as the `graphicx` package is.

`bb=`*a b c d*	*Bounding box* (§29)
`scale=`*s*	Scaling factor
`angle=`α	Angle of rotation (between -360 and 360 degrees).
`origin=`*p*	Origin of rotation (§31).
`width=`*w*	Desired width for the image (§30).
`height=`*h*	Desired height for the image (§30).
`totalheight=`*h*	To use if the image is rotated more than 90° or clockwise (§30).
`keepaspectratio`	To keep the original width/height ration (§30).
`viewport=`*a b c d*	Similar to `bb`, used to see *only* the portion of the image between a rectangle of vertices (a, b) and (c, d). It should be used in conjunction with `clip` to prevent the rest of the image from being printed.
`trim=`*a b c d*	Reduces the bounding box (§29) by the specified quantities.
`clip`	Used with `viewport` to clip the image.
`draft`	The image is not printed but a box of the corresponding size replaces it (processing time is shorter).

Table 1: Options for `\includegraphics`.

§29. *The bounding box* of a graphic is information on its size that LaTeX needs to allocate the necessary space in the page. It should be given in the scheme

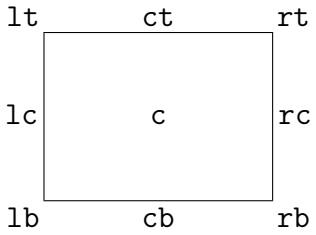


Encapsulated PostScript graphics (`.eps`) have the bounding box information embedded within the files, so when importing them there is no need to tell LaTeX the four numbers a, b, c, d (of course, using EPS in PDF files is another problem, addressed below in §32ff.). But for other

graphic formats (possible formats depend on the implementation and the driver, but in general such standard formats as JPG, TIFF, and PNG can be included) the bounding box has to be specified. Almost every graphic editor can give this information, sometimes in the form of 'pixel size,' in whose case $(a, b) = (0, 0)$.

§30. *The size of the graphic* can be controlled with options `scale`, `width`, `height`, and `totalheight`. The latter is recommended when a rotation of more than 90 degrees (or of a negative quantity) is made to the figure. The other options mentioned are self-explanatory; `scale` takes a number, `width` and `height` take a LaTeX dimension. If only one of `width` and `height` is given, the other is accommodated to keep the ration. Giving both will probably deform the image, unless `keepaspectratio` is explicitly requested.

§31. *Rotation of graphics* is achieved with the options `angle` and `origin`. The latter sets the point of the image around which the rotation is made, and the available values are the following:



PostScript graphics in PDF documents

§32. *The incompatibility between PostScript and PDF* formats would seem, at least to the ignorant, one of the biggest fiascos of modern computing, being as it is that both languages were developed by Acrobat. At any rate, what this essential incompatibility implies in LaTeXnique terms is a serious corollary: nothing (no packages, no commands, no nothing) that contains '`ps`' in its name can be trusted to work properly for PDF files. **PSTricks** and `\psfig`, just to name two, are big losses.

Many tools have been written to overcome this difficulty. Perl scripts, GhostScript routines, etc., are available from CTAN.[6] Packages exist with suggesting names like `pdftricks` (an adaptation of **PSTricks** for PDFLaTeX) or `ps4pdf`. My own lack of experience with any of them makes me unable to recommend anything more substantial than 'go and check them out' (and then let me know!).

In part, this lack of experience is due to my opinion that the efforts to install and actually run these programs (many of them assume a Unix system, or otherwise limit their range of action) are not really worthwhile. There are very much simpler solutions that are quite enough for the average user's graphic needs.

§33. There is, for example, a good first way to try to include *external PostScript files*, namely the dvipdfm program (that I strongly recommend as the best way to create PDF files from LaTeX input). Suppose a LaTeX document imports PostScript graphics. If the document is processed through PDFLaTeX, the graphics will not show up. But running LaTeX on it will create a DVI with the graphics (that are internally embedded by means of `dvips`, a virtually standard program nowadays). Then, applying dvipdfm on this DVI will create the PDF, and, what is more relevant, dvipdfm will automatically try to convert the graphics, through its own GhostScript routine, and embed them into the PDF file. Normally it will succeed (although there is always the possibility that it cannot convert the file, if the latter contains some command or construction unknown to it).

For all this to be possible, of course, the system has to have `dvips`, dvipdfm, and GhostScript installed. Also, if the LaTeX document is created with pittetd, option `dvipdfm` should be declared (see pittetd's manual for more on this and related options).

The PostScript files for this method have to be external. This includes most uses of METAPOST, but fails to include **PSTricks**, that creates *internal* PostScript graphics.[7]

[6] 'Comprehensive TeX Archive Network,' an on-line archive with everything about TeX and LaTeX. Most its contents is public domain. The site can be accessed from `http://www.tug.org`, the TeX Users Group's webpage.

[7] In addition, it should be noted that PDF TeX (and PDF LaTeX, by extension) has support for METAPOST.

§34. *Internally generated PostScript* output, which is what **PSTricks** does, is harder to manage. It seems a good idea to generate the images in a separate LaTeX document, creating a DVI file, then to convert it to `.ps` (with `dvips`) and include it as a regular external file (§33). But somehow it does not work. It surely has to do with the fact that **PSTricks** does not understand 'drivers' (§27), so it has no way to accommodate to dvipdfm's conventions (nor to PDF TeX's, for that matter). At any rate, other LaTeX tools that employ PostScript handling, such as the `graphicx` package itself, are well understood by both dvipdfm and PDF TeX.

In any case, the alternative is to convert the internally-created PostScript file to other formats that can be included.

§35. *Converting PostScript files* is done through external programs. Acrobat Distiller, or Acrobat PDF Writer (a 'printer emulator' that 'prints' into PDF files) can create a PDF file of the image. Then, if the driver to the `graphicx` package is `pdftex` (§27) and the document is to be processed with PDF LaTeX, the new PDF file can be imported with `\includegraphics` (§28ff.).

Or else, the file can be fitted in the screen, 'captured' (the screen is copied to the clipboard), and pasted on a new graphic file—of any format, in any graphics program. (Again, which formats can be imported depends on the local system and the driver used, see §27.) The new file is then handled appropriately with the `\includegraphics` command (§28ff.).

Indexing with *MakeIndex*

The enormous task of compiling an index is facilitated—to a certain extent—by the program *MakeIndex* and the LaTeX package `makeidx`, written by Phong Chen and Nelson Beebe, and included in standard LaTeX distributions. A guide to the program [3] is available from the Pitt ETD Webpage.

§36. *The process*, similar to that of bibliographies generation with BibTeX, is as follows:

- A first LaTeX run compiles the relevant information, namely the `\index` commands, and writes it into an external auxiliary file with extension `.idx`.
- The program *MakeIndex* formats and sorts out that information, producing another file, of extension `.ind` this time.
- A second LaTeX run reads the latter file and typesets the index into the document.

§37. *Preparation of the input file*. The input file has to be 'prepared' to generate an index. First, the next two lines have to be present in the preamble of the document:

```
\usepackage{makeidx}
\makeindex
```

`\printindex` Second, the index itself has to be requested by typing

```
\printindex
```

where it is desired. After these preparations, LaTeX is ready to interpret `\index` commands (on which below). A normal run—something like `latex doc.tex` in the command line—produces an additional `doc.idx` file; running *MakeIndex*—something like `makeindex doc`—and then LaTeX again will create the index.

`\index` §38. *Indexing terms* is done through the command `\index{`⟨*term*⟩`}`. One such command is needed for each indexed term, and should be issued right after the term itself (avoiding spaces that could result in page breaks and wrong pagination). For example:

```
The most evident manifestation of the light\index{light}
as \emph{particles}\index{particles} (i.e.\ as
photons\index{photons}) occurs at the screen. The light
arrives there in discrete localized units of
energy\index{energy}, this energy being invariably related
to the light's frequency\index{frequency} in accordance
with Planck's\index{Planck, Max} formula: $E=h\nu$.
```

As can be seen, the `\index` commands are quite intrusive—the input file becomes difficult to read. That is a good reason why the

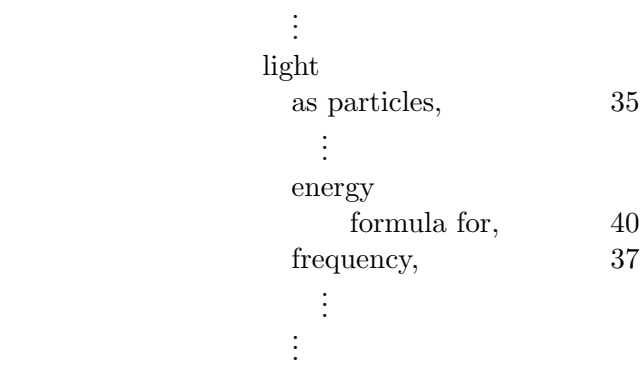
index should be compiled when the text itself is in its final form, or very close.

Many aspects of the use of `\index` are detailed in the following paragraphs. Switches and parameters for *MakeIndex*'s command line are presented starting with §46. §47 presents a package that helps tracking indexed terms.

§39. *Sub-terms*. `\index` can create subdivisions for the terms in the index, indicated by the `!` character:

```
\index{light!as particles}
\index{light!frequency}
\index{light!energy!formula for}
```

The above commands (issued, say, in pages 35, 37, and 40, respectively) would create something like



There can only be three levels in the index (sub-sub-terms).

§40. *Multiple references*, i.e., indexing the same concept under several index entries, can be done by successive `\index` commands. Spaces between `\index` commands should be avoided, and a good way to do this is using the `%` character. For example:

```
The most evident manifestation of light as particles%
    \index{light!particles}%
    \index{photons}%
occurs at the screen.
```

§41. *Alphabetization* is automatically made by *MakeIndex*. In doing so, it is sensible to both case and space. In other words, each of

`\index{␣set}`, `\index{set}`, `\index{Set}`, and `\index{SET}` create different items in the index.

You can override the automatic alphabetization of any term, using the `@` character to tell *MakeIndex* where the term should go:

`\index{`⟨*position*⟩`@`⟨*term*⟩`}`

For example, `\index{pi@$\pi$}` would place the symbol π in the position of 'pi,' instead of that of `$`. This is also useful for expressions like '`\emph{MakeIndex}`' or '`\"Agler`', which would ordinarily show up before any letter ('alphabetized' as starting with `\`).

The modifier can also be used in the subterms part of `\index` (§39). Thus, `\index{quarks!colors@‘colors’ of}` will place the subterm under the letter c rather than the character ‘.

§42. *Italic and bold page numbers* are sometimes given different meanings in the index (the present document, for example, italicizes the page number where the entry is described in most detail). This is done with the character `|` within the `\index` command, as in
`\index{indexation|textit}`

This command, if issued in page 36, would generate:

⋮
indexation, *36*
⋮

Note that in these constructions the character '`|`' substitutes the '`\`' of the regular commands. Other such 'commands' can be used, such as `|textbf` or `|underline`.

§43. *Page ranges* for an indexed term (for example, when it is the subject of an entire section, so that the index entry should show something like '13–15') are specified by `||`

`\index{`⟨*term*⟩`|(}`	at the beginning of the range,
`\index{`⟨*term*⟩`|)}`	at the end.

The numbers can be further formatted in the opening `\index`, for example in `\index{`⟨*term*⟩`|(textit}` (which is still to be closed regularly by `\index{`⟨*term*⟩`|)}`).

§44. *Cross references* with the expression '*see*' are produced by the 'command' `|see`, as in

```
\index{light quanta|see{photons}}
```

which creates an index entry like

⋮
light quanta, *see* photons
⋮

The similar `|seealso` writes '*see also*' instead of '*see*'.

§45. *Special characters* `@`, `!`, and `|` have a special meaning for *MakeIndex*, and therefore they cannot be used freely within `\index` commands. If you need one of them to appear explicitly in an entry, it has to be preceded by `"`, as in `\index{nobody"@nowhere.com.nw}` or in `\index{x@$"|x"|$}`.

§46. *Running* MakeIndex. *MakeIndex* is a binary program that in its simplest form takes the document's file name (with extension `.idx`, although this can be omitted) as a parameter. Optional switches can be appended, some of which are explained below. In general, then, *MakeIndex* is executed (on the command line) by

`makeindex` ⟨*switches*⟩ ⟨*filename*⟩

The most important ⟨*switches*⟩ are:

`-c` makes *MakeIndex* ignore spaces within `\index` commands. If this switch is used, `\index{set␣}` will be alphabetized identically with `\index{set}`.

`-l` makes *MakeIndex* alphabetize multiple-word expressions with no regard for middle spaces. `prime numbers` will be alphabetized as `primenumbers`.

-r disables the formation of page ranges. Instead of '24–26', *MakeIndex* will produce '24, 25, 26'.

§47. *Tracking indexed terms* is made easier with the package `showidx`, that makes `\index`'ed terms visible, just as the `showkeys` packages

makes cross-referencing keys visible (§19). When `showidx` is loaded (`\usepackage{showidx}`), each page will feature a list of the terms that are indexed in it.

The pittetd class has a patch for this file (whose only purpose is to avoid a series of `overfull box` messages), so it should be loaded with `\usewithpatch`.

§48. *Hyper-indices with* ***`hyperref`***. The `hyperref` package, used by pittetd to create links and bookmarks, offers the possibility of making an interactive index, where the page numbers are links to the corresponding pages. However, this is not very robust: it virtually disables the use of the modifiers that follow the `|` character within `\index` commands, so that the features described in §42–§44 are disabled. That is why pittetd disables interactivity for the index, although this can be overridden (see section 6.2 of pittetd's documentation).

A The `\acro` command

Here are the lines that provide the `\acro` command (§4) for standard classes:[8]

```
\makeatletter
\DeclareRobustCommand\SMC{%
  \ifx\@currsize\normalsize\small\else
   \ifx\@currsize\small\footnotesize\else
    \ifx\@currsize\footnotesize\scriptsize\else
     \ifx\@currsize\large\normalsize\else
      \ifx\@currsize\Large\large\else
       \ifx\@currsize\LARGE\Large\else
        \ifx\@currsize\scriptsize\tiny\else
         \ifx\@currsize\tiny\tiny\else
          \ifx\@currsize\huge\LARGE\else
           \ifx\@currsize\Huge\huge\else
            \small\SMC@unknown@warning
 \fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
}
```

[8] This piece of code comes from the ltugboat class.

```
\newcommand\SMC@unknown@warning{%
    \PackageWarning{acro}{\string\SMC: unrecognized
    text font size command -- using \string\small}}
\newcommand\textSMC[1]{{\SMC #1}}
\newcommand\acro[1]{\textSMC{#1}\@}
\makeatother
```

B Illustration of `\lips`

The following is a comparison between the results of `\dots` and `\lips` (§5). It comes from the `lips` package's documentation.

`Hello\dots.␣And`	Hello. . . . And
`Hello\lips.␣And`	Hello. . . . And
`Hello␣\dots.␣And`	Hello . . . . And
`Hello␣\lips.␣And`	Hello. . . . And
`Hello.\dots␣And`	Hello. . . . And
`Hello.\lips␣And`	Hello. . . . And
`Hello.␣\dots␣And`	Hello. . . . And
`Hello.␣\lips␣And`	Hello. . . . And
`one,\dots,␣three`	one,. . . , three
`one,\lips,␣three`	one, . . . , three
`one,␣\dots,␣three`	one, . . . , three
`one,␣\lips,␣three`	one, . . . , three
`Hello\dots␣and`	Hello. . . and
`Hello\lips␣and`	Hello . . . and
`Hello␣\dots␣and`	Hello . . . and
`Hello␣\lips␣and`	Hello . . . and
`Hello!\dots␣And`	Hello!. . . And
`Hello!\lips␣And`	Hello! . . . And
`Hello!␣\dots␣And`	Hello! . . . And
`Hello!␣\lips␣And`	Hello! . . . And

Glossary

Comment out When LaTeX sees a `%` character, it ignores the rest of the line. This is useful for making personal 'comments' to the input file, but also to (temporarily) 'delete' things. This is what

is referred to as 'commenting out.'

Counters LaTeX counters are integer variables. The page number, the chapter number, and the number of floating objects in a page, for example, are stored by LaTeX in counters.

A special kind of counters have an effect in cross referencing, and can be retrieved by a `\ref` command, when a `\label` has been assigned to them. These are the counters for chapters and sections, figures, tables, footnotes, and `enumerate` items.

New counters can be created with `\newcounter{`⟨*name*⟩`}`, and modified (as can be also the pre-defined counters) by typing `\setcounter{`⟨*name*⟩`}{`⟨*new value*⟩`}`, `\stepcounter{`⟨*name*⟩`}`, and `\refstepcounter{`⟨*name*⟩`}`. The latter makes the counter affect cross-references.

Declarations Commands that, rather than affect an argument, change the general behavior from the moment of their occurrence on. Typically used in the form `{\declaration ...}` (rather than `\declaration{...}`.) Font size commands, for example, are declarations.

Dimensions See 'LaTeX Dimensions.'

LaTeX dimensions LaTeX understands many dimensional units, among them `in`, `cm`, `mm`, `pt` (`1in` = `72pt`), `em` (the width of an 'M'), `ex` (the height of an 'x'). When a 'dimension' is asked for, the natural answer is to give it in terms of these units. But dimensions can also be given in terms of previously defined 'lengths,' such as `\textwidth`, `\parindent`, etc.

LaTeX lengths Commands that hold a length. Many lengths are defined by LaTeX itself. Others can be created by the user with `\newlength{`⟨*command name*⟩`}`, and all can be managed with `\setlength{`⟨*command name*⟩`}{`⟨*LaTeX dimension*⟩`}` and `\addtolength` (same syntax).

Packages Additions to LaTeX that offer new commands or features. To be used they have to be loaded, which is done by `\usepackage[`⟨*options*⟩`]{`⟨*package*⟩`}`

The part [⟨*options*⟩] is of course optional. When using pittetd, packages can (and maybe should) be loaded with the command `\usewithpatch` instead; this will make pittetd look for a 'patch' that solves any incompatibility issues (see pittetd's manual).

Preamble The part of the input file between `\documentclass` and `\begin{document}`. All packages are loaded and some parameters are set within the preamble.

References

[1] Rodrigo De Castro, *El Universo LaTeX*, 2nd. edition, Bogotá, Universidad Nacional de Colombia, 2003.

[2] Leslie Lamport, *LaTeX: A document preparation system*. Addison-Wesley, Reading, Massachusetts, second edition, 1994.

[3] _____, MakeIndex*: An index processor for LaTeX*, February 17, 1987. File `makeindex.tex` (`dvi` output available at the Pitt ETD Webpage).

[4] Oren Patashnik, BibTeX*ing*, February 8, 1988. File `btxdoc.tex`, documentation to BibTeX version 0.99b (`dvi` output available at the Pitt ETD Webpage).

Index

The index is by paragraph (§) instead of page. Numbers in italics refer to the paragraph where the corresponding entry is described.