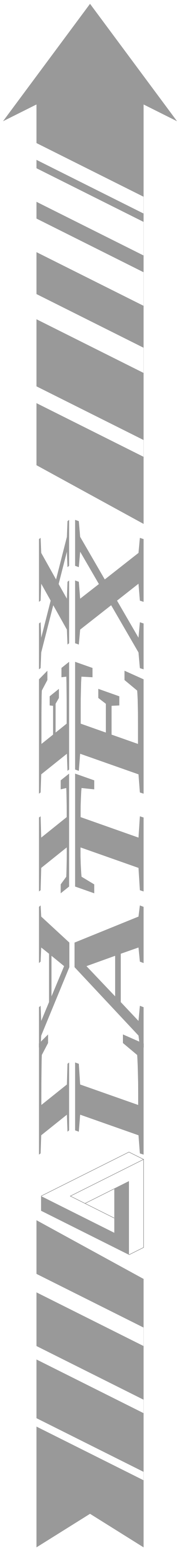
LameTeX

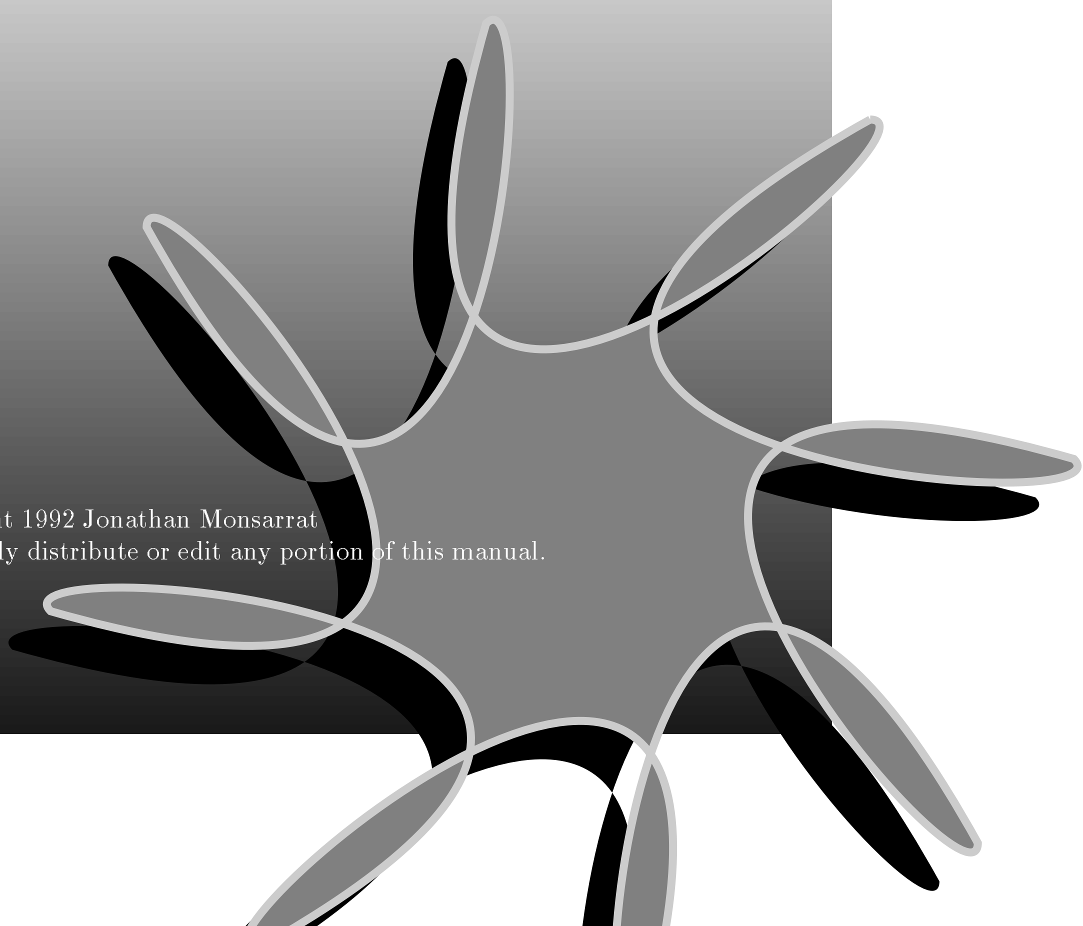


LameTeX

A text formatter for special effects

Combining the magic of PostScript with the ease of LaTeX

Version 1.1 by Jonathan Monsarrat
September 8, 1992



Chapter 1

Getting to Know LameTeX

Hackers, just skip to the section “Fast Track”.

1.1 Copyright

Don’t sell LameTeX or this documentation, or use parts of it in programs you sell. Don’t even think about selling it. This isn’t about money. It’s about cool hacking. Permission granted for everything else. Thank you.

1.2 About Text Formatters

A *text formatter* is a program that helps you make nice looking reports and documents. You write the document by typing the text into an editor, adding formatting commands as needed. A *formatting command* is not meant to be printed. Instead it is interpreted by the text formatter program to accomplish special tricks such as centering a line of text, that you would otherwise have to do by laboriously typing the space bar a number of times.

LameTeX is a text formatting program that understands many of the most important text formatting commands of a popular text formatter, LaTeX. Although not as powerful for normal text formatting, LameTeX has a number of additional features for special effects.

1.3 LameTeX is for Special Effects

That said, if you want to write a plain report or do some fancy mathematics, you should use LaTeX instead. LaTeX does a better job for normal-looking documents (this page is an example of a normal-looking LaTeX page), and LameTeX version 1.0 is not capable of handling some of the more esoteric LaTeX functionality like mathematics and command definition. That’s why it’s called “Lame” TeX!

1.3.1 What LameTeX Can Do

LameTeX’s specialty is complete versatility of the printed page. The standard model for text formatters is that every page is necessarily rectangular. LameTeX will let you format text inside a triangular page, or a circle page. Just like professional magazine editors, you can include pictures of *any* shape and ask the text to flow around them or inside them.

These flexible arbitrarily-shaped margins are PostScript paths. If you don’t know PostScript, there is a big library of interesting LameTeX page margins here. With LameTeX you can fit several "pages" onto one 8.5x11 inch piece of paper, so you can easily make index cards, labels, and half-pages of text.

Also, if you know how to write programs in PostScript, LameTeX allows you to very tightly integrate your LameTeX commands with your PostScript code. In fact, the PostScript that LameTeX outputs is nicely formatted and commented so that you can modify it yourself and see how it's done. LameTeX is written with PostScript version 1, so it should run on all PostScript printers.

Finally, everything about LameTeX is set up to be compatible with LaTeX. LameTeX can't do everything that LaTeX can, but the special *stealth commands* guarantee that your fancy LameTeX document can be processed by normal LaTeX. This allows you to share it with anyone who doesn't happen to have LameTeX. (Better yet: give them a copy; LameTeX is free!)

1.3.2 What you Need to Know to Learn LameTeX

LameTeX is a tool to make things simple. You don't have to be a LaTeX expert to use LameTeX, but you should be somewhat familiar with some basic LaTeX commands before reading this document.

You don't need to know any PostScript to use LameTeX, although there is a lot extra you can do with LameTeX if you can do just a little.

You *definitely* don't need to know anything about TeX to use LameTeX.

LameTeX is written in C++ and PostScript from scratch. It is not based on TeX macros or anything like that. It just happens to know the same language as LaTeX, and also it snarfs fonts from LaTeX.

1.4 The Fast Track

If it's 3 a.m. and you are a juggling, tetris-addicted hacker with a taste for reheated Chinese take-out food, you may want to skip to the good parts. Look for the Little Wizard icon for interesting notes for experts who won't need to read this whole manual.

- The LameTeX code for this document, `lametex_doc.tex`, and all the files it includes, are good examples of what LameTeX code looks like. Read through it and you will notice it's not hard to make a few special effects. In general, just try lots of LaTeX commands, and LameTeX will either handle them or attempt to skip them gracefully.

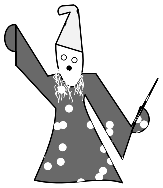
- Look in the appendix to see what commands LameTeX supports if in doubt. **Warning! LameTeX lets you write sloppy code.** Try writing your document in LaTeX first, to work the bugs out of the "normal look" and then start using LameTeX. Then you're guaranteed that your document will work on both text formatters.

- Also glance at the section on Stealth Commands, which are LameTeX commands over and above the normal LaTeX commands. The stealth command `ignore` tells LameTeX to ignore LaTeX commands it doesn't know and can't skip gracefully.

- LameTeX is written in C++, and the code itself is documented separately.

- The PostScript routine for breaking a PostScript path into tiles is breakpath.ps. And the formatting language is handled by format.ps. You might find these programs useful outside of LameTeX. For example, I wrote a 3D drawing environment which used breakpath.ps for tiling.

- There is an appendix at the back on installing LameTeX.
- The author is a nice guy who will be happy to help you out. Send e-mail to `jgm@cs.brown.edu`. I would like to know what you think of LameTeX and if you found it useful. I am also accumulating a list of enhancements to make to the program and would gratefully receive your ideas on this subject.



1.5 Major LaTeX Commands LameTeX Cannot Handle

The most obvious difference between the two programs is that LameTeX will never hyphenate words, and will tend to not have quite as nice spacing as LaTeX does. LameTeX can only produce PostScript code, and the code it does produce is complicated and therefore a little long.

LameTeX uses the LaTeX fonts and handles the computer modern font robustly, but it cannot currently handle other font families or normal PostScript fonts. You cannot use fancy non-ascii characters like the heart and the copyright symbol.

You cannot define new commands in LameTeX, make boxes around words, or use fancy document formats like "titlepage", or "letter". You cannot do fancy tabbing or horizontal spacing. Math mode is right out.

A lot of new functionality will be added in the next version of LameTeX coming late this year, but for the moment keep in mind that LameTeX is meant primarily for placing special effects on rather basic documents. Use normal LaTeX, or cut and paste between the two, if you need math mode or want to nest lots of TeX code in your LaTeX files.

Chapter 2

Simple Formatting LameTeX

This chapter introduces to the essentials of running the LameTeX program, but is primarily an overview of LaTeX buzzwords and syntax. Those grossly familiar with LaTeX can just glimpse at the little wizards and skip to the next section.

2.6 Running LameTeX

2.6.3 Useful Buzzwords To Know

A *LaTeX file* is a text file full of the words of your document, plus some fancy formatting commands. The LaTeX text formatter knows how to interpret the formatting commands from your LaTeX file and produce a printed result. The filename of a LaTeX file usually ends with the suffix ".tex" but this is not required.

A *LameTeX file* is just like a LaTeX file, except that LameTeX has some extra text formatting commands that LaTeX doesn't know. Also, LameTeX doesn't know all of the commands that LaTeX knows. Fortunately, for the basic commands, both programs work fine. For reference, there is a complete list of all text formatting commands that LameTeX understands at the end of this document.

LaTeX will understand any well-written LameTeX document because the fancy LameTeX commands are "hidden" inside comments. Any LameTeX command that LaTeX doesn't know is called a *stealth command*.

A big part of LameTeX is being able to make your own very fancy page descriptions. A *page description file* is a file full of some PostScript page descriptions, written in the PostScript programming language.

2.6.4 Example 1 - A Plain Vanilla Text File

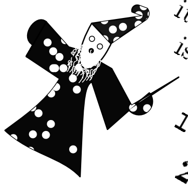
LameTeX comes with a number of example LameTeX files. They all begin with the prefix "example", then a number, and end with the conventional suffix ".tex".

Try formatting the LameTeX file `example1.tex` by typing **`lametex example1.tex`**. LameTeX will process the file, converting it to the graphics language PostScript. Your screen should look something like this:

```
vegas %lametex example.tex
This is LameTeX, C++ Version pre 1.0
Processing example1.tex...
Opening example1.PS for temporary output...
Including PostScript file /home/jgm/PS/Lametex/page/page_
latex.ps
Making dummy file for snarfing LaTeX fonts...
Snarfing LaTeX fonts...
This is TeX, C Version 3.0
(lametex.tex
LaTeX Version 2.09 (24 May
1989)
```

```
(/cs/lib/tex/inputs/report.sty
Document Style ‘report’, (13 Nov 89).
(/cs/lib/tex/inputs/rep10.sty)
(/cs/lib/tex/inputs/titlepage.sty))
(/cs/lib/tex/inputs/10pt.sty) (lametex.aux) [ 1 ]
(lametex.aux) )
Output written on lametex.dvi (1 page, 344 bytes).
Transcript written on lametex.log.
This is dvips 5.47 Copyright 1986-91 Radical Eye Software
’ TeX output 1992.08.22:0325’ - lametex.ps
(tex.pro). [ 1 ]
** SNARF! **
Created PostScript file example1.ps
vegas %
```

LameTeX has produced a new file, **example1.ps** that you can print on any PostScript printer using the **lpr** command. Alternatively, you can view it on-line by using GhostScript or some other PostScript previewer. It is recommended that you *always* use a PostScript previewer before printing, just to make sure that you get what you want.



As part of the text formatting process, LameTeX must acquire the correct “official” LaTeX fonts, and to do this, it makes a small trojan horse file and tricks LaTeX into interpreting it. The dummy file it creates is *lametex.tex* which is formatted by LaTeX to become *lametex.ps*. All temporary files are deleted after they’re used.

You are welcome to omit the suffix *.tex* if you wish, just as with LaTeX. For example, you could have typed **lametex example1**.

2.6.5 Specifying the Default Page Description File

The *default page description* is the PostScript file that is used to decide what the page should look like, if you don’t specify a page description. You can use the -p command line option to specify a new page description. Try typing **lametex -p page_skew.ps example1.ps** to format the same example LameTeX file with a different page description. You can use any of the page descriptions in the LameTeX page description directory.

If you don’t specify the default page description, it automatically defaults to **page_latex.ps**.

You can use the -t command to produce plain ASCII text output. Just type **lametex -t yourfile.tex** and the result will be placed in yourfile.txt.

2.7 A Brief Review of LaTeX

Following is a brief review of what normal LaTeX commands look like. You should peek at a real LaTeX manual if this section confuses you.

If you formatted **example1.tex**, you saw that the printed result was placed so that it lined up on both columns. Each new paragraph was indented, and the title was centered on the page.

Take a look at **example1.tex**, and you will see that the plain LameTeX file is not nearly so pretty! LameTeX was responsible for doing the formatting. LameTeX shares responsibility for decision-making with you, the user. For example, there is a special notation that makes LameTeX center the title of the document. A normal LameTeX document is full of such decision-making commands, either to ask for something new, or to override a LameTeX decision (like whether to indent a paragraph).

2.7.6 \documentstyle

The most important command in this example is **\documentstyle{report**
}.

LaTeX wants to know what kind of document you are trying to produce. LameTeX, it turns out, doesn’t really care! All document styles look the same to LameTeX, but unfortunately you must have the\documentstyle command anyway, just because that is the way the cruel world is sometimes. Plus, you need it if you are ever going to run your document through normal LaTeX. There are three alternate versions of this command, `\documentstyle [ 10pt ]{report}`, `\documentstyle [ 11pt ]{report}`, and `\documentstyle [ 12pt ]{report}`, which specify that the default font size for this document will be 10 point, 11 point, or 12 point respectively. The real LaTeX text formatter can do a lot of other fancy things with the \documentstyle command, but this version of LameTeX cannot.

2.7.7 \begin and \end Delimit Environments

An *environment* is an important LaTeX concept. It tells LaTeX where to start and stop doing fancy stuff. For example, in the example LameTeX file **`example1.tex`** there are two commands, `\begin{center }` and `\end{center}`. The \begin command says "start doing a fancy thing, which is to center every line.". Likewise, the \end command is very important because it tells LaTeX where to **stop** centering. We say that the text in-between the \begin and \end command is "in a \center environment".

A shorthand notation for an environment is to use braces. For example, the line of text

`\begin{center } this text is centered\end{center}`

could be re-written as follows:

`{\center this text is centered}`

2.7.8 Text Goes in the \document Environment

Whenever you use LaTeX, you must place any text to be printed inside of a \document environment. Don't place text outside of a \document environment, although you may place text formatting commands there.

The two commands you will *always* want to make part of your LameTeX documents are \documentstyle and \document.

2.7.9 Example 2 - Environments Can Be Fancy

Try running the second example file, by typing **`lametex example2.tex`**. This LameTeX file doesn't have any stealth commands either. Like **`example1.tex`**, it is just a demonstration that LameTeX can handle some pretty complicated normal LaTeX text formatting commands.

This file contains some fancy environments. For example, the \verse command modifies the environment in several ways to allow a natural look to poems.

The \enumerate environment indents the text and numbers all paragraphs that begin with the \item command.

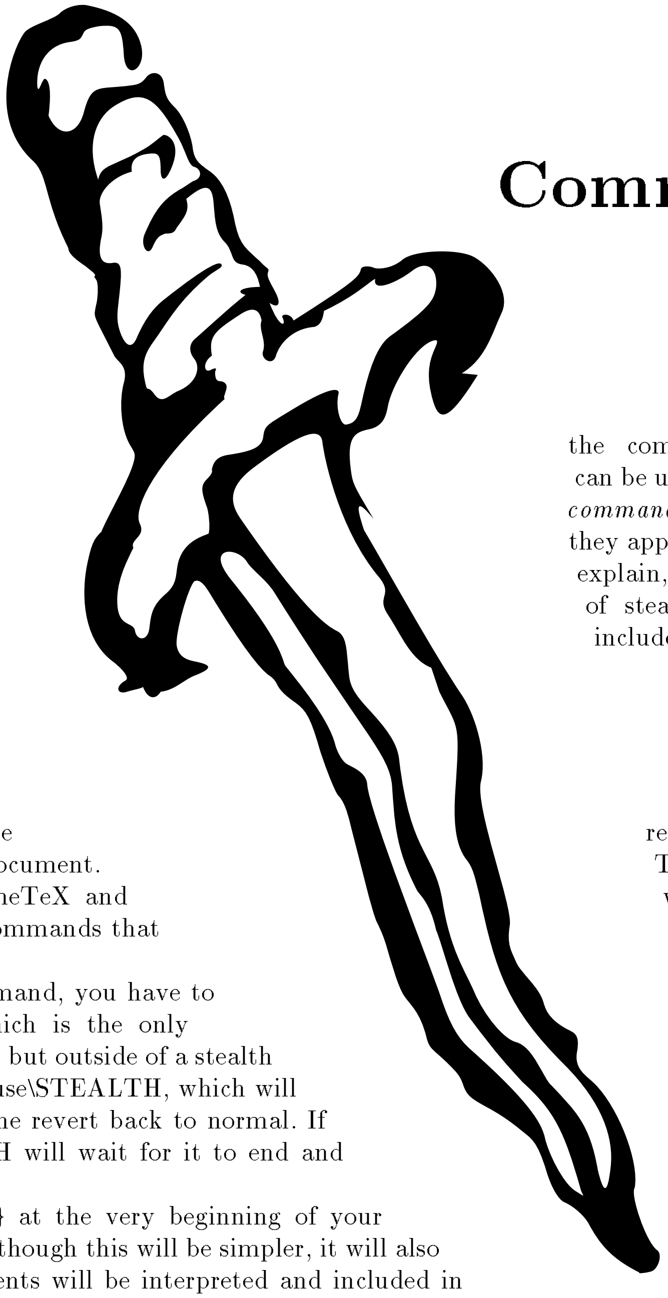
2.7.10 Environments Can Be Nested

Also notice that the \enumerate commands can be nested inside one another. Also the \bf command for **boldface** is nested inside the \Huge command for Huge text. Putting the boldface environment inside the Huge environment yields text that is **both bold and Huge**.

In general, when you nest environments, "it does the right thing." In general you can trust LaTeX to *do the right thing*. Can you trust LameTeX? Well ... you know the old saying: "Trust your mother. But watch her!"

Chapter 3

Stealth Commands



LameTeX doesn't understand all the commands that LaTeX does, but it does know a few extra that can be used for special graphics. These commands are called *stealth commands*, and are used like any other LaTeX command, except that they appear inside comments!

Stealth commands are difficult to explain, but easy to demonstrate. Try looking for examples of stealth commands inside `lametex_doc.tex` and the files it includes.

3.8 \begin{stealth}

Normally it would be nice if you could make real comments without LameTeX inserting the text into your document. This is especially true if you are converting a LaTeX file to LameTeX and want to make extensive use of the comment symbol to take out commands that LameTeX cannot handle.

So, in order to begin a use a stealth command, you have to use a stealth environment by using\begin{stealth}, which is the only command that is interpreted by LameTeX inside a comment but outside of a stealth environment. You cannot just say\stealth. If you want that, use\STEALTH, which will continue the stealth environment to the end of the line, and the revert back to normal. If you begin a\ postcript environment then the\STEALTH will wait for it to end and then terminate on the next newline.

You are welcome to do a\begin{stealth } at the very beginning of your document and end it with\end{stealth } at the end, but although this will be simpler, it will also mean that all text (including plain text) in your comments will be interpreted and included in your document. So to avoid confusion and mistakes, and to allow you to use real comment, it is best if you actually begin a local stealth environment only when needed, and close it immediately afterwards.

The stealth command is useful when you have some plain text that you don't want normally included in LaTeX. For example, in this document whenever it says to "look for the little Wizard icons" found in the LameTeX version, the text referring to the icons is imbedded in a stealth environment in comments, so that normal LaTeX will skip the text.

Warning! The stealth command does not really start a delimited environment. For example, if you were to have

`\ begin{Huge }\bf FOO\end{Huge } BAR`

then the FOO would be Huge and boldface, but the BAR would be neither. This is because the Huge environment serves as a delimiter for everything inside of it.

The stealth command is different. Unlike every other begin/end combination, if you were to say

`%begin{stealth }\bf FOO\end{stealth } BAR`

then FOO would be boldface, and BAR would also be boldface.

3.9 Including a PostScript file

You can use the\includeps command to include a PostScript file. For example, to include a file of PostScript definitions in **gummi.ps** you would type the stealth command `\ includeps{ gummi.ps}`.

3.10 Choosing a Page Description

You can use the\pspage command to include a new page description. See the chapter on pages to learn more about this command.

3.11 For Fancy LaTeX Stuff

You can tell LameTeX to “skip over” some part of your file in which you might, for example, imbed LaTeX commands that LameTeX cannot handle or skip gracefully. You must say\begin{ignore } and use\end{ignore } inside a stealth environment to do this enclosure.

3.12 Free text PostScript

You can type any PostScript commands you want directly by using the\ postscript command. For example, to change the gray scale of the printed text, you can type `\ begin{postscript } .6 setgray\end{postscript}`.
You **cannot** just say\postscript.
There are a lot of special things to learn about exactly where a postscript command (like a grayscale) that you supply will be applied.

3.13 Hacks Just Like Mom Used To Make

Here’s a fancier way to get your PostScript commands executed exactly where you want them to. Use the\pscmd command to specify a PostScript function to run, and the function will be run exactly in the text where you have specified.

You would use this command only inside a line of text. Between lines of text, just using the\postscript environment is appropriate.

Everything you type in a PostScript environment gets dumped to PostScript, except for the first comment character ’ you want to make a comment in your PostScript, make sure to use two ’

Read the chapter on pages to learn how to use this “direct stranglehold” on PostScript correctly -- unless you enjoy surprises!

Chapter 4

How Pages are Handled in LameTeX

If you're going to use the fancy pages included with this release of LameTeX, but not try to design your own or make any PostScript hacks, then you don't need to read about LameTeX pages. However, if you are going to try your own hand at designing pages and fancy graphics, you should learn a little bit about how LameTeX works so that things fit together smoothly.

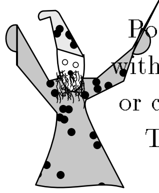
I'd love to hear what you've used LameTeX for. Please send e-mail to `jgm@cs.brown.edu`.

4.14 Behind the Scenes of LameTeX

4.14.11 The Page Description

A Page Description is a short program written in PostScript that defines some standard stuff about the page that text gets placed on.

Take a look at `page_latex.ps`, which is the standard plain vanilla LaTeX-looking page. Here are the routines defined in this file:



PageShape is a **required** routine that defines a PostScript path for the page. A path is basically a squiggle on the page that never gets drawn. It is a closed loop of arbitrary shape. It is pretty easy to make a simple path out of straight lines using the PostScript `moveto` and `lineto` commands, but it is possible to define a very complex path with curves and fancy wiggles also. You can read more about paths in the PostScript blue book, or chapter 4 of the red book.

The PageShape routine should leave a number on the stack called a *setflat value*. If your path contains fancy curves, it is going to get flattened into a series of lines using the PostScript **flattenpath** operator. Usually "currentflat 8 mul" works fine, although if you want more preciseness and think your printer can handle it, try smaller values. If you don't understand this, just use “currenflat 8 mul” and you should be fine.

StartPage is a **required** routine that draws anything or sets up anything required at the beginning of a page. Feel free to put some fancy graphics here.

EndPage is a **required** routine that draws anything or sets up anything required at the end of the page. For example, the one in `page_latex.ps` prints

InitPage is not required. It is just a simple enclosure for defining some variables that *are* all required:

fillout tells LameTeX whether you want to place text *inside* the defined path, or *outside* of it. If you set fillout to true LameTeX will not place any text inside your path.

evenodd is a fancy PostScript way of determining what is inside and what is outside of a path. If this is true then it uses the evenodd rules; false means use the winding rule. There's a description of what the heck this means in the PostScript red book in Chapter 4, section 6. Basically setting this variable to true means that every part of the path delimits an outside-inside boundary.

BM In the event that fillout is true, this is the bottom of the bounding rectangle that will be filled with text. If parts of the path extend below this given number, it will be reset to be below the path.

TM In the event that fillout is true, this is the top of the bounding rectangle that will be filled with text. If parts of the path extend above this given number, it will be reset to be above the path.

LM In the event that fillout is true, this is the left of the bounding rectangle that will be filled with text. If parts of the path extend to the left of this given number, it will be reset to be to the left of the path.

RM In the event that fillout is true, this is the right of the bounding rectangle that will be filled with text. If parts of the path extend to the right of this given number, it will be reset to be to the right of the path.

LeftMarginIcon is not required. I just use it do define where the left margin is, in case I want to place an Icon there.

4.14.12 The Page Cycle

LameTeX has three modes that it can be in:

1. In-Between Pages
2. On a Page, In-Between Lines
3. On a Page, On a Line

The cycle generally goes like this:

1. LameTeX starts off in “In-Between Pages” mode.

2. In-Between Pages mode processes commands until it encounters any plain text or LameTeX command that might require that a new page be started. When this happens it does the following:

    3. Execute the `save` command

        (a) Start a page by calling the /StartPage routine.

        (b) Do some internal initialization and call the /PageShape routine

        (c) Change to "On a Page, In-Between Lines" Mode.

        (d) On a Page, In-Between Lines mode processes commands until it encounters any plain text or LameTeX command that might require that a new line be started. When this happens it does the following:

            i. Change to "On a Page, On a Line" mode.

            ii. Find a new line. NOTE: If you have broken the page up into two or more horizontal regions (like `page_dagger.ps` does, then know that a "newline" really means a new "open space to put words". It's possible to have "two new lines" horizontally next to one another. One goes on the left side of the dagger page description. The next new line starts on the right side at the same horizontal position. If you don't like this functionality, then don't define regions that have two sides like the `page_dagger.ps`. Instead define "two pages" like how `page_check.ps` works. The first page will get filled before the second is begun.

            iii. Initialize a word list that will contain information about this line. The very first item on this word list will be a command to change to what is considered the proper font when this line was begun.

            iv. If appropriate for each new word or command read in, it will either add the word to the word list or add the *command itself* to the word list. For example, a boldface command in between the words FOO and BAR gets inserted in the word list between FOO and BAR. It also get executed immediately.

            v. Also for each new word read in, it checks to see if the word list it is building overflows the width of the current line (as defined in a very fancy way by the PostScript Page Definition file and the PostScript path given by /PageShape). If there is an overflow, or a natural end of line given by a text formatting command, then it does the following:

                A. Go through the word list from the beginning to the end and for each element, either print it if it is a word, or execute it if it is a command. Since the word list was initialized with a font-setting command, the first thing this loop will do is choose the proper font. Add a little space between words if text is supposed to be fully justified.

      B. If the cause of printing this line was a natural end of line, then go to step 3.3.

      C. Otherwise, create a new word list, and keep cycling in “On a Page, On a Line” mode.

   (e) Eventually, going to a new line means falling off of the bottom of the page, and a we change back to “In-Between pages mode”, and do the following:

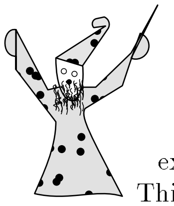
4. Execute the `restore` command

5. End a Page by calling the /EndPage routine.

6. Go to step 2.

LameTeX horizontally aligns the words by remembering all the words on the current line, and then printing the whole line at once when the line gets filled up. Printing all the words at once means that it can be smart enough to justify the text to exactly match both margins.

There is no concept of vertical alignment. Unlike LaTeX, LameTeX does **not** remember all the lines in a paragraph and then print the paragraph all at once in the right place.



The **save** and **restore** commands are PostScript operators to do memory management. Basically what happens is that when the `restore` command is encountered, the entire current memory of the printer is replaced with what it was like when the `save` command was executed.

This means that if you set a variable in your StartPage routine, you will be able to access that variable throughout the length of the page, but you will not be able to access it in the EndPage routine. If you want to have some always-changing always-remembered variable like the page number, use the InitPage and EndPage routines instead.

Note that the InitPage and EndPage routines are *outside* of the save-restore loop. So anything that happens in these two routines stays around forever.

Which Commands Force A New Page Or A New Line?

Check it out in `Operator.C`. Yes, that’s a C++ source file. Don’t be afraid of it - it won’t hurt you. In this file at the top is a huge table of command names. The first row is the command name, the second is a boolean (either a 0 or a 1) saying whether this command is a stealth command. The third is a boolean (either a 0 or 1) saying whether or not a new page or new line should be started if this command is executed In-Between Pages or Not-On-A-Line.

Also, all plain text words force On-A-Line-ness and On-A-Page-ness.

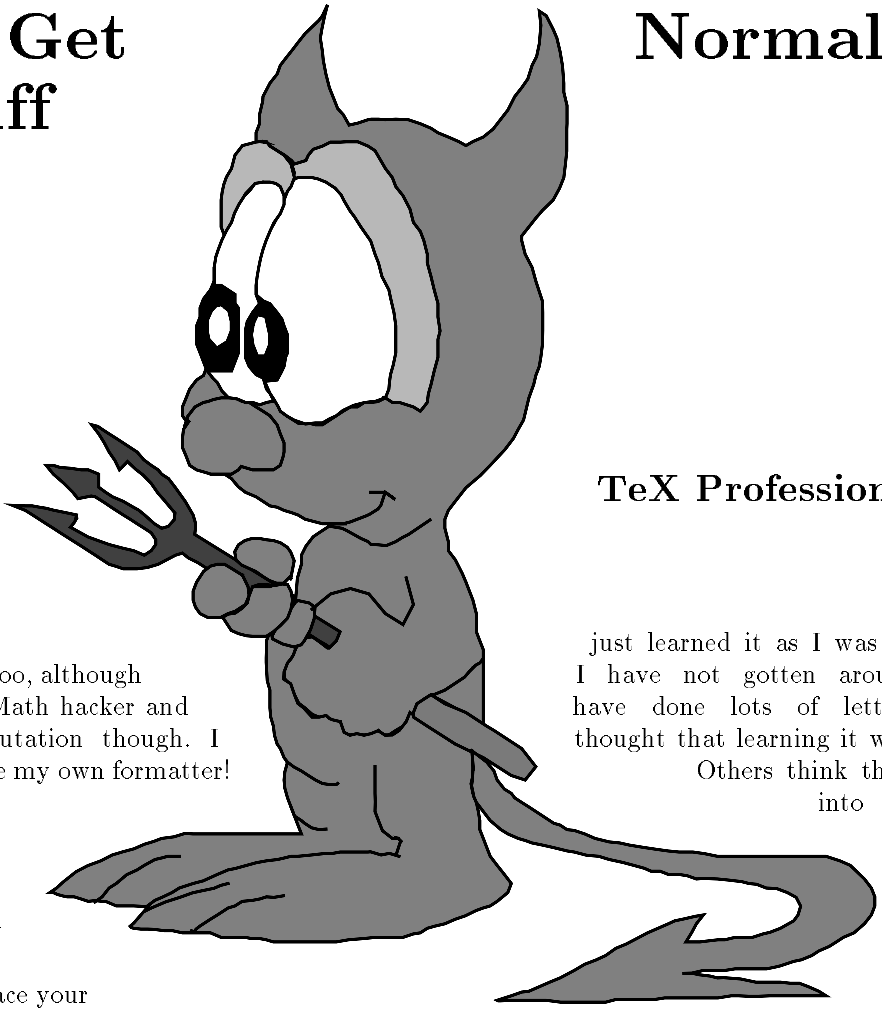
Specifying the Page Description Directories

LameTeX will look in the current directory to find page description files or any other postscript files, but if it cannot find them you must tell it where to look. You can use the -p command, explained in the startup chapter, or set your environment variable LAMETEX_PS_PATH to some path which will be searched for postscript files. The current directory and the LameTeX main library directory will always be searched.

There is a great deal more about this in the PostScript chapter.

Chapter 5

How to Get Normal LaTeX Stuff



5.15 A Note to TeX Professionals

I love LaTeX. It's great. I just learned it as I was writing LameTeX. I'm sure TeX is cool too, although I have not gotten around to learning it yet. I was a Scribe Math hacker and have done lots of letterpress printing. TeX has a bad reputation though. I thought that learning it would be so hard that I might as well write my own formatter! Others think that TeX kind of straight-jackets you into professional documents.

Normally that's great. My Mom isn't around any more to clean my room and tell me how to write professionally. And a letterpress doesn't care if you place your letters upside down.

But occasionally (or *often* if you're strange like me) I get the urge to splash 100 point type or stick random bugs all over the page. Once to advertise an event I hung up a huge 18 foot poster taped together out of PostScript pages. You ever get that urge?

Anyway, LameTeX is a fun toy. Just don't take it too seriously.

And no, it's not written in TeX macros!

5.16 The LaTeX Length Parameters

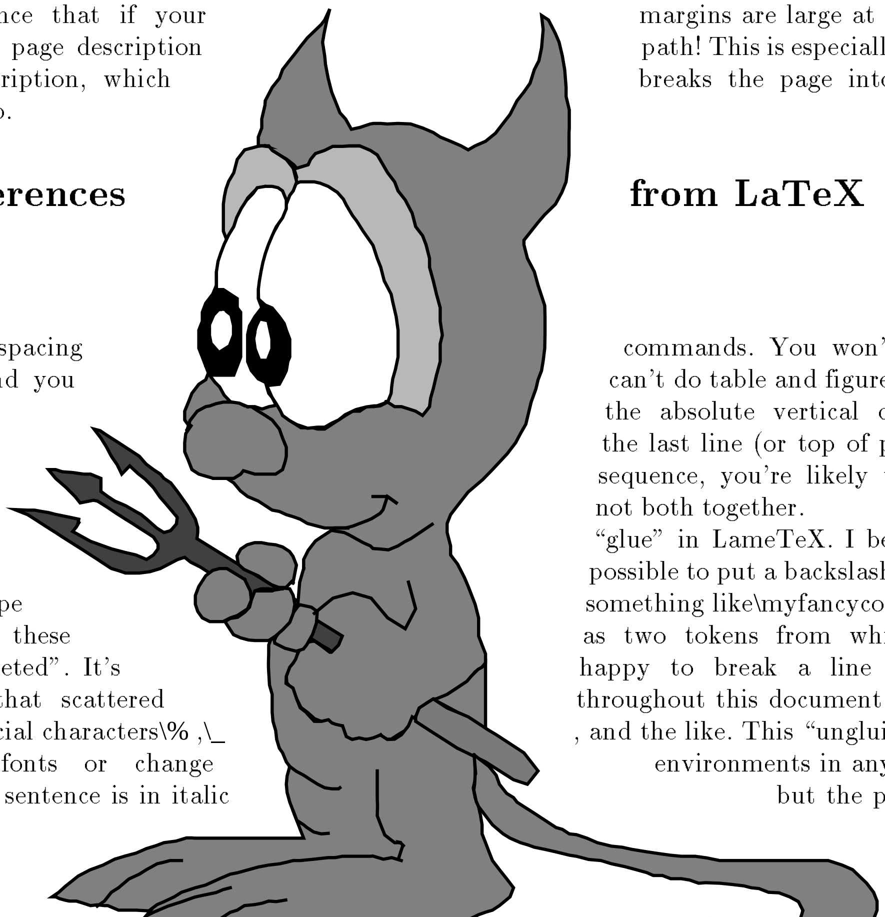
If LaTeX is royalty, then LameTeX is a prostitute. If you want to change the margins *in the middle of the page* using\textwidth, LaTeX kind of sticks its nose up at you and says to go implement your own margin library font program hierarchy system out of TeX macros. LameTeX just says "OKAY!" and hops in the sack with you. No questions.

So feel free to change a lot of stuff. LameTeX won't care. I really urge you to put random stuff inside of a stealth environment, however, so that normal LaTeX won't balk at it and refuse to serve you.

It's like playing an adventure game. If you die, who cares, it's just a program. But if you don't try various things, you won't have any fun.

There is no "textwidth" in LameTeX. When you set textwidth it just sets the right margin to 8.5 inches minus the textwidth given minus the left margin.

Similarly there is no "textheight". When you set textheight it just sets the bottom margin to 11 inches minus the textheight given minus the top margin.



Please note that just like LaTeX, LameTeX defaults to ridiculous margins. There's a good chance that if your margins are large at all, that the text may not fit inside the page description path! This is especially true of the **`page_check.ps`** page description, which breaks the page into boxes that are only and inch wide or so.

5.17 Tiny Differences from LaTeX

You can't use fine-tune spacing commands. You won't get a table of contents anywhere, and you can't do table and figures.

\vspace is meant to give the absolute vertical distance between the current line and the last line (or top of page). If you use two of them in a sequence, you're likely to have the larger one apply only, not both together.

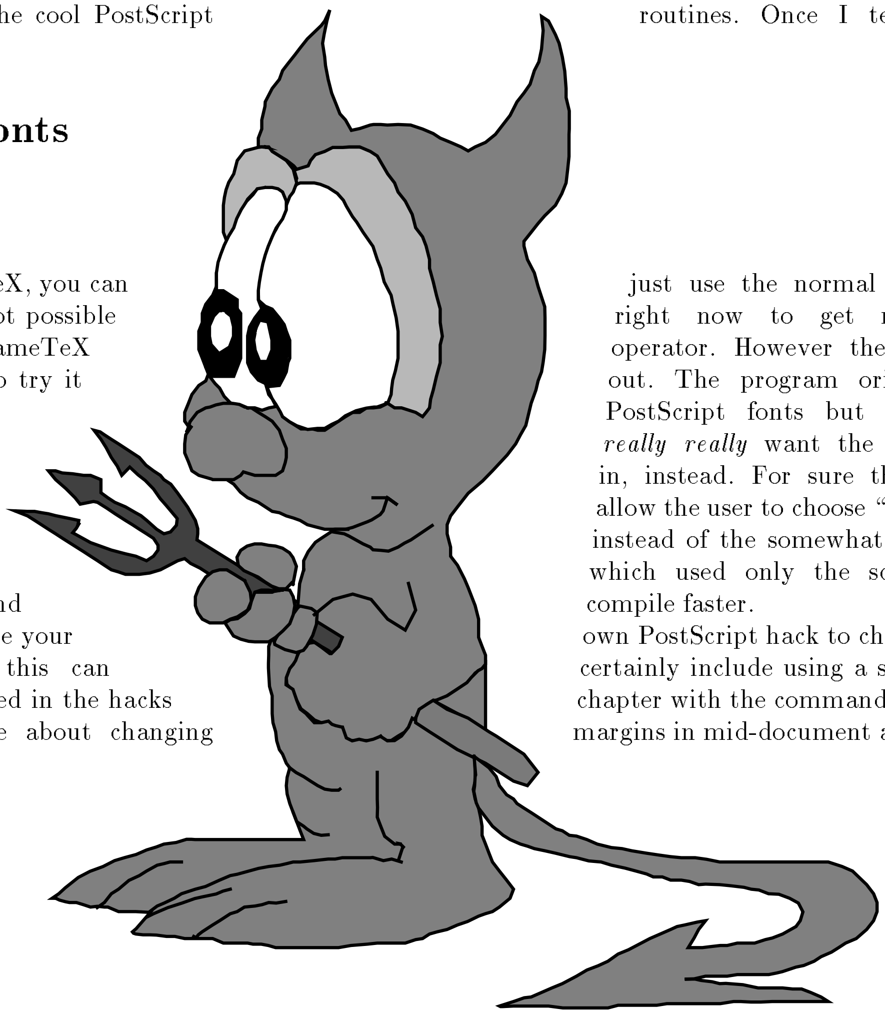
There is no concept of "glue" in LameTeX. I bent over backwards trying to make it possible to put a backslash next to a command when I want to type something like\myfancycommand. Unfortunately, LameTeX views these as two tokens from which the space in-between has been "deleted". It's happy to break a line in the middle. You'll probably see that scattered throughout this document. It only happens when you print the special characters\% ,_ , and the like. This "ungluing" will also occur if you change fonts or change environments in any way in the middle of a word. So if my sentence is in italic but the period at the end of the sentence is in roman type, the period might be placed on a newline by itself!
In general, LameTeX will occasionally screw up the spacing.
Oops. No, it doesn't do kerning! Geez
...
To avoid this it's best to include sentence punctuation in whatever environment happens to precede it. For example, use "Pay me in {\em cash.}" instead of "Pay me in {\em cash}."

You can't make small caps, typewriter, or sans serif font into italic or bold face. I didn't know it was possible until recently so I didn't put it into the program.

LameTeX is quite happy to give you widows and orphans. If this sort of thing keeps you awake at night, you are taking LameTeX too seriously and should go play Tetris until you go insane (this doesn't take long). If you don't know what widows and orphans are, good. I won't tell you.

5.18 PostScript is a Great Language, Too Bad it Sux

I love PostScript. LameTeX makes extensive use of the PostScript programming language. I think PostScript is great. However, when a PostScript program breaks, your printer is more likely to say “ARGH” or silently start whining rather than to print out some message like “I say, old man. You’ve got a spot of trouble on line 36 you see. It seems (if I may venture to take a liberty) that you made a misspelling error in the word ’sux’.” If it’s a LameTeX bug then get in touch with me and I’ll try to fix it. But if you’re hacking PostScript through LameTeX it’s very easy to screw things up. The solution is not to get scared and stop playing with it. Send me some mail and we’ll work it out together. Chances are I forget to document some assumption I’m making somewhere about how to use the cool PostScript routines. Once I tell you about it things will be fine.



5.19 LameTeX Fonts

To get different fonts in LameTeX, you can just use the normal LaTeX font-changing operators. It is not possible right now to get normal PostScript fonts using a LameTeX operator. However there is a cool workaround if you want to try it out. The program originally worked that way on PostScript fonts but then I figured that people would *really really* want the normal LaTeX fonts, so I put those in, instead. For sure the next version of LameTeX will allow the user to choose “normal” scaleable PostScript fonts instead of the somewhat tedious LaTeX bitmaps. Documents which used only the scaleable fonts would be much smaller and compile faster.

It is possible for you to make your own PostScript hack to change the appearance of a word, and this can certainly include using a scaleable font if you wish. This is described in the hacks chapter with the command\pscmd.

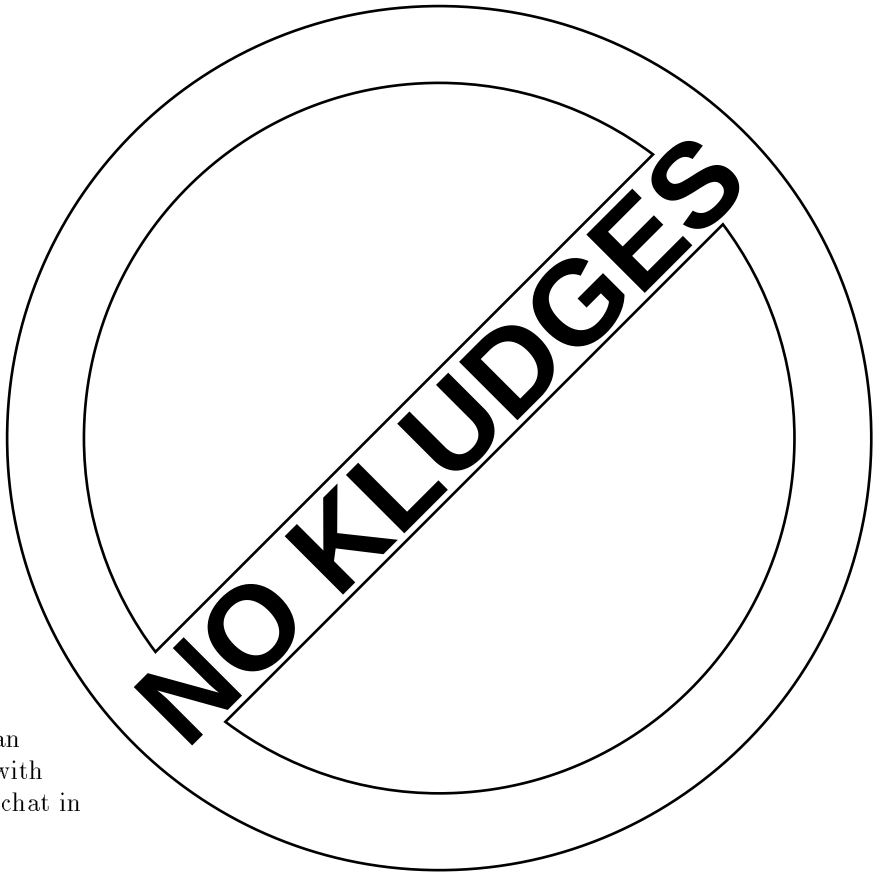
LameTeX is very versatile about changing margins in mid-document and stuff like that.

5.20 Getting References

LameTeX allows you to use the LaTeX commands\ref and\label, but not\pageref. If you have forward references, you do *not* need to run LameTeX twice. LameTeX is smart enough to do its own recovery if you have forward references. Pretty neat, huh?

Chapter 6

Including Your Own PostScript Hacks



Let's face it. This manual is nice,
but let's get into the nitty gritty of the
examples presented here and maybe I can
show you how to do some cool stuff with
LameTeX. If you have questions, we can chat in
person over mail. Here goes.

6.21 How to Use Include Files To Do Stuff

This manual can be formatted by running either LaTeX or LameTeX on the file `lametex_doc.tex`. There are a lot of other files, but `lametex_doc.tex` includes them using the LaTeX command\include.

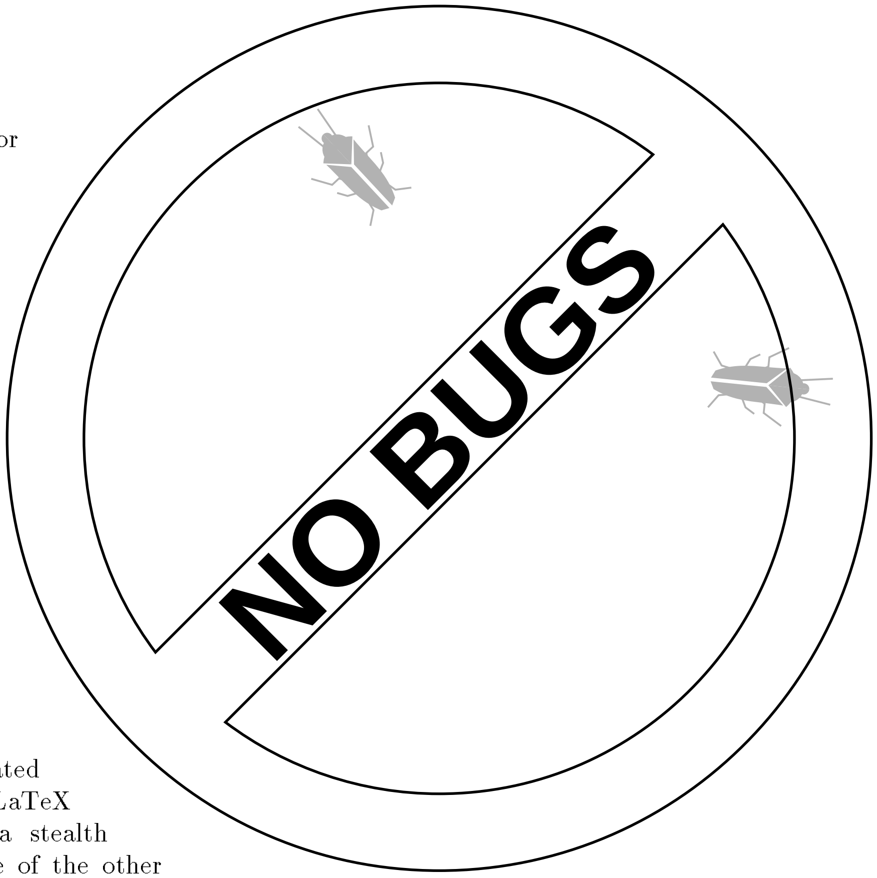
While reading this section on examples, you will probably want to run LameTeX over just one chapter, not all of them. If you edit the file `lametex_doc.tex` you can "comment out" everything you don't want. For example, if you wanted to run LameTeX over the chapter on pages, you could edit the file so that it looked like:

```
\documentstyle [ 10pt ]{report }
\setlength{\textwidth}{6 in }
\setlength{\textheight}{9 in }
\setlength{\topmargin}{0 in }
\setlength{\oddsidemargin}{0.25 in }
```

```
%\begin{stealth }\includeps{icons.ps }\end{stealth }
    \begin{document }
     %\include{titlepage }
     %\include{introduction }
     %\include{starting }
     %\include{stealth }
    \include{pages }
     %\include{normality }
     %\include{hacks }
     %\include{future }
     %\include{errors }
     %\include{commands }
    \end{document }
```

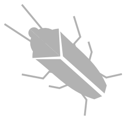
Notice how all the chapters, except for pages.tex, have been placed in comments using the '%' character. This means that both LaTeX and LameTeX will ignore them.

6.22 White Text Hack



Look in titlepage.tex, just before where it says "Copyright 1992". Here is a demonstration of changing the color of the text so that it is white instead of black. This is done using the special LameTeX stealth commands\stealth and\postscript.

All the stealth commands are located inside comments so that the normal LaTeX won't see them. So I have to start a stealth environment, because without that none of the other commands I type inside the comments will be interpreted. Then I start a postscript environment. From that point on, everything in the file gets dumped directly to the postscript output. So "1 setgray" gets included directly in the PostScript output file. This is the PostScript command to paint things white, not black.



Naturally after the one line has been painted white, I must put the default color back to black or else my whole document will be in white! The\stealth and\postscript environments are special because they don't as scope delimiters for internal changes.

So if you did a boldface in a Huge environment you'd expect that when the Huge environment ended, that the boldface would go away. The "scope" of boldface was limited to the Huge environment.\stealth and\postscript don't limit scope like that.

Here's a trick to try out. Try placing the words "Copyright 1992" before the "1 setgray" line and the words "Jonathan Monsarrat" afterwards. What happens? Does "Copyright 1992" come out in white or black?

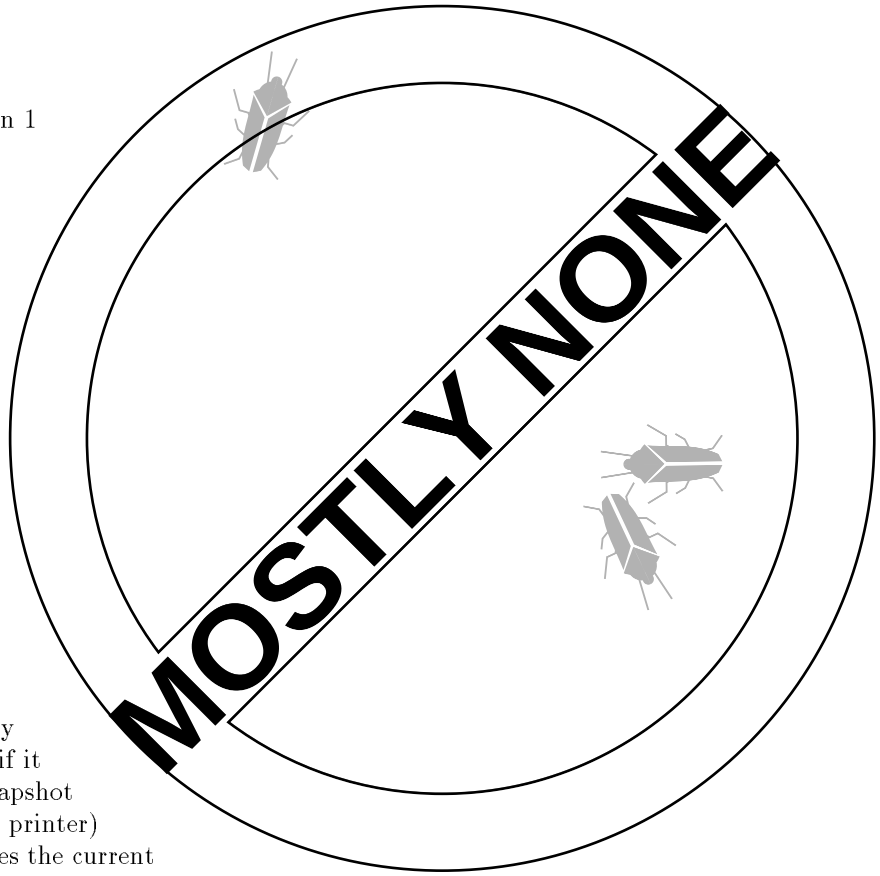
The answer is that is comes out in white, and if you refer to the Page chapter I can show you why. LameTeX is trying to center this line on the page. To do that it needs to know how long this line is. So instead of printing out the words one at a time, LameTeX stores the words in a big list, and then prints them at the end once the line has been completed. LameTeX always does this.

So in the modified version I asked you to make, first a procedure to get the correct font is put on the word list.

Then “Copyright” is added to the word list. Then “1992” is added. So far, nothing has been printed. Then the “1 setgray” gets executed. Then “Jonathan” and “Monsarrat” get added to the word list. Finally we reach the end of the line, and when the whole line prints out, all four words are in white because of the “1 setgray” command.

The word list, as well as having words and commands on it, can also have numbers. A number on a word list means to move over an amount horizontaly. The\hspace command and others add in horizontal whitespace in this way.

6.23 The Dagger Hack



LameTeX is written for PostScript version 1 because I wanted it to work on all PostScript printers. I love PostScript but it was very awkward to work with. For one thing, PostScript 1 doesn’t have very good memory management tools. If you’re not careful, even a simple hack like piping a thousand words through a small program and printing them will take up a lot of memory, because each word is stored in a string, and that string is never deleted, not even when the string is printed and cannot be accessed.

Fortunately, there is a workaround. LameTeX uses the PostScript commands `save` and `restore` to do memory cleanup. If it didn’t, it would use up so much memory that your document could not print out if it were long. The `save` command takes a snapshot of the current state of computer (or printer) memory and the `restore` command erases the current state and returns the computer (or printer) to exactly the state it was in whe the last `save` command was executed. This happens at the beginning and end of each page, as mentioned in the chapter on Pages.

Now here’s an interesting question for you. I have this immense picture of a dagger that I want to include in my PostScript page. I want text to flow around the dagger. What should I do?

Look in `stealth.tex` and `page_dagger.ps` to find the answer. If I were to place the huge Dagger routine outside of a save-restore block, then it would hang around in memory forever, even if I didn’t intend to use that particular picture any more in my document. This would be unclean. What I really want to do is to insert the Dagger picture, including the definition - because the definition takes up a lot of room too as well as variables - inside the StartPage routine.

The StartPage routine gets called after the save and before the restore. So whatever garbage I throw into memory drawing the dagger will neatly get erased at the end of the page.

Also there is the matter of the path defined in PageShape. The dagger itself is some 200 lines of curveto commands! This is way too much curviness and way too detailed for something as macroscopic as placing text. So I have drawn an irregular polygon around the dagger. The polygon works just as well for the simple blocky task of placing text around the dagger. Plus it is a lot simpler because it is made entirely of straight lines.

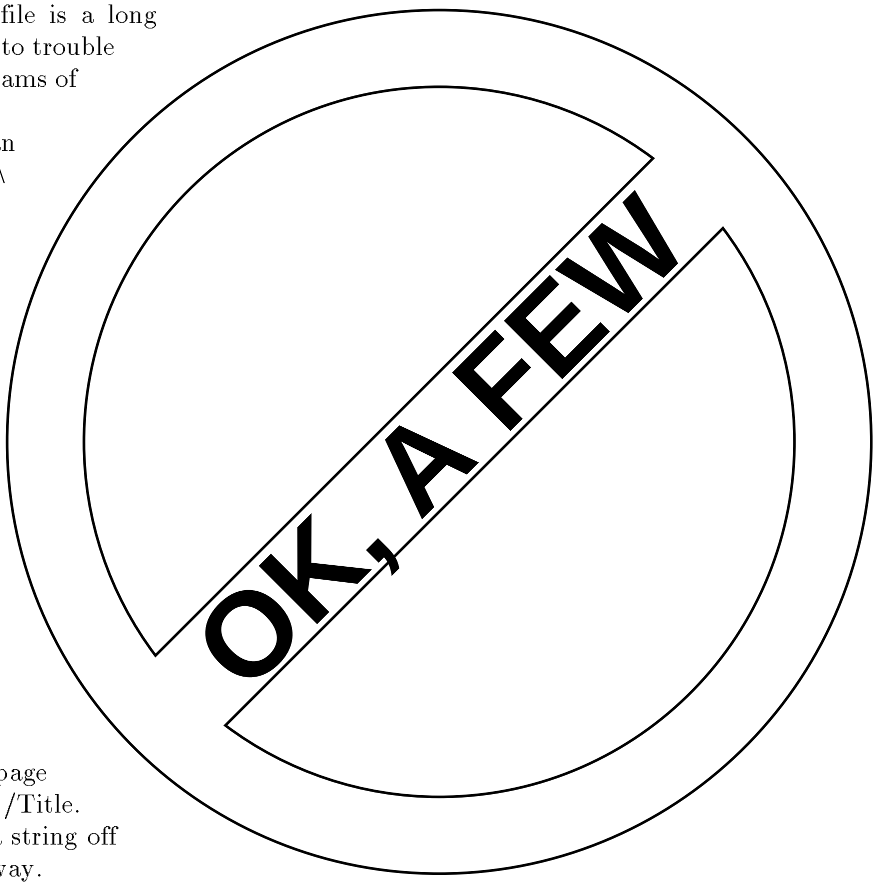
You can view this path by uncommenting the line where it says “Uncomment me” in `page_dagger.ps`.

Actually the description of the dagger itself gets thrown into the In-Between Page space that is not cleaned up, because the StartPage routine is defined along with the page itself in the In-Between Page space.

One way to get around this I have implemented for `page_demon.ps`. That is to have the page description just contain a small bounding box, just like the description for the dagger. The actual drawing of the fancy demon I include in normality.tex, once on each of the three pages. Why would I include the file three times? Doesn't that make my PostScript file bigger?

Yes, that is certainly true. But the demon hack is immense, some 25K of raw PostScript code. I really don't want to define a PostScript routine to draw all of that. If I use the include function, the routines that draw the get demon are executed without ever being saved. So it's OK to have a very big PostScript file as long as the file is a long stream of commands. You start getting into trouble when you define procedures with long streams of commands.

Unfortunately, this does indeed mean that I have to "guess" where to place the\ include demon.ps command, once for each of the three pages.



6.24 The Titlebar Hack

The best thing about LameTeX is that if you want to do some pretty radical stuff, LameTeX says "OK! Let's go for it!", whereas TeX and LaTeX would probably say "B-B-But, why would you want to do that? That wouldn't be... (trumpet flare) *professional!*"

Here's a great example. On the title page there are two big hacks, /Weird-Stuff and /Title. Both of these PostScript programs take a string off of the stack and display it in an unusual way.

Using these two programs is as simple as starting a stealth environment, starting a postscript environment, and then just inserting the commands!

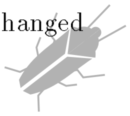
NOTE: Notice how the LameTeX title has holes in it! An omen? A sign of portent about the program itself?

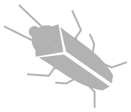
Also note that of course I put all of the major graphics coding into the StartPage routine, including the definitions, so that when they are no longer needed these definitions will be erased. The definitions persist between the save and restore, and since all the printed text on the page comes between the save and restore, I can still call these routines from the LameTeX document as long as I do it before the restore command associated with the EndPage.

6.25 Page Number Hack

This hack is the logical opposite of the dagger hack. With the dagger hack, I wanted to stuff a lot of PostScript code into the /StartPage routine, get it to draw some stuff, and then destroy it.

With the page number, I want the page number to be preserved across pages. It would be a crime if I changed



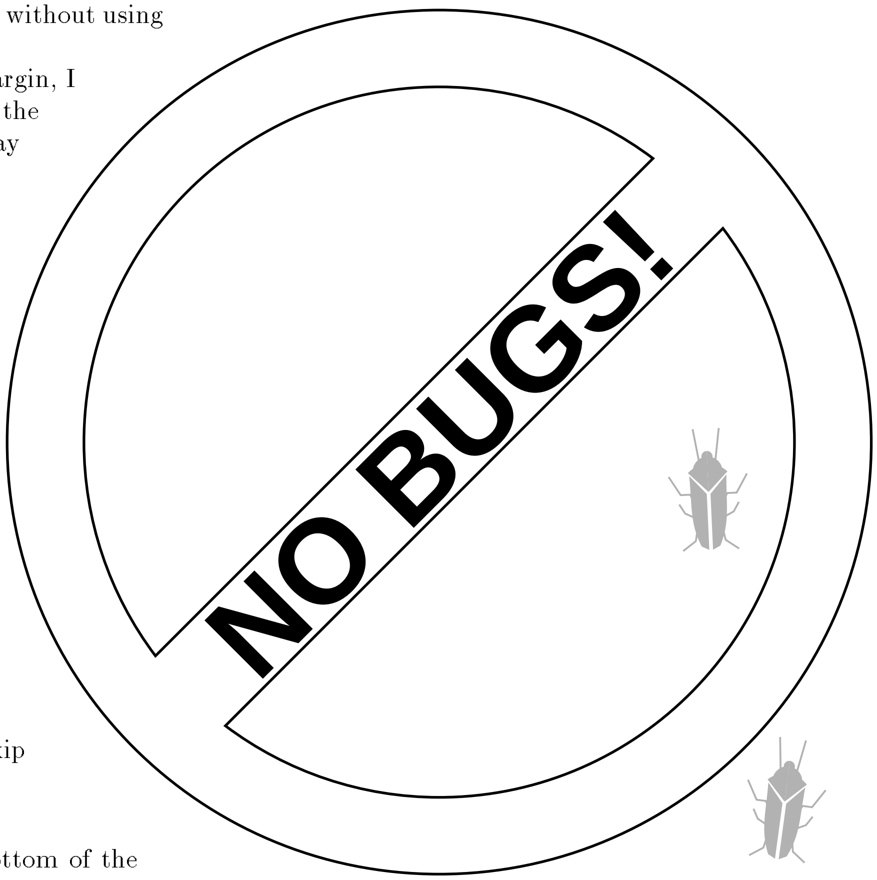


page number 2 to 3, and then when the restore command came along it reverted back to 2! So instead of placing this code in /StartPage, it goes in /EndPage instead.

The /EndPage routine gets called *after* the restore from the previous page and *before* the save from the next page. It's in-between pages, and any variables changed here, like pagenumber, will remain around. Also the InitPage routine gets called before the first save is done for the page, so it is in-between pages also.

Here's another quirk to inbetween-pages. When you're on a page, a special dictionary is used called formatdict. A dictionary is just a place to store variables in PostScript. These variables are accessed directly like normal variables when you're on a page. But in-between, you need a special way to get at the variables. One exception: InitPage does have access to the variables directly, which is how it sets formatdict variables like BM and evenodd without using dictionaries.

For example, if I want the bottommargin, I can't just say "bottommargin" to put the bottommargin onto the stack. I have to say "formatdict /bottommargin get", which does the same thing. If you are a PostScript expert, you'll know what I say when I note that I do this to keep plenty of room open in the generic user dictionary.

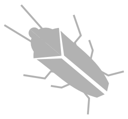


6.26 The Format Dictionary

The most important variables in the format dictionary are:

baselineskip The number of lines to skip between lines.

bottommargin The margin at the bottom of the page. The LaTeX words textheight and textwidth don't mean anything to LameTeX. They get translated into left, right, top, and bottom margins on the assumption that the textheight and textwidth commands apply to a normal-sized 8.5x11 inch paper.



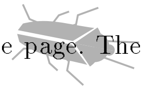
bslot The X coordinate of the left hand side of the current line.

coords A very complex representation of the page path.

eslot The X coordinate of the right hand side of the current line.

justify An integer, can be 102 (or 'f') for full justify, 99 (or 'c') for centered, 114 (or 'r') for flush right, or 108 (or 'l') for flush left.

leftmargin The margin at the left hand side of the page. There is no distinction between evensidemargin and



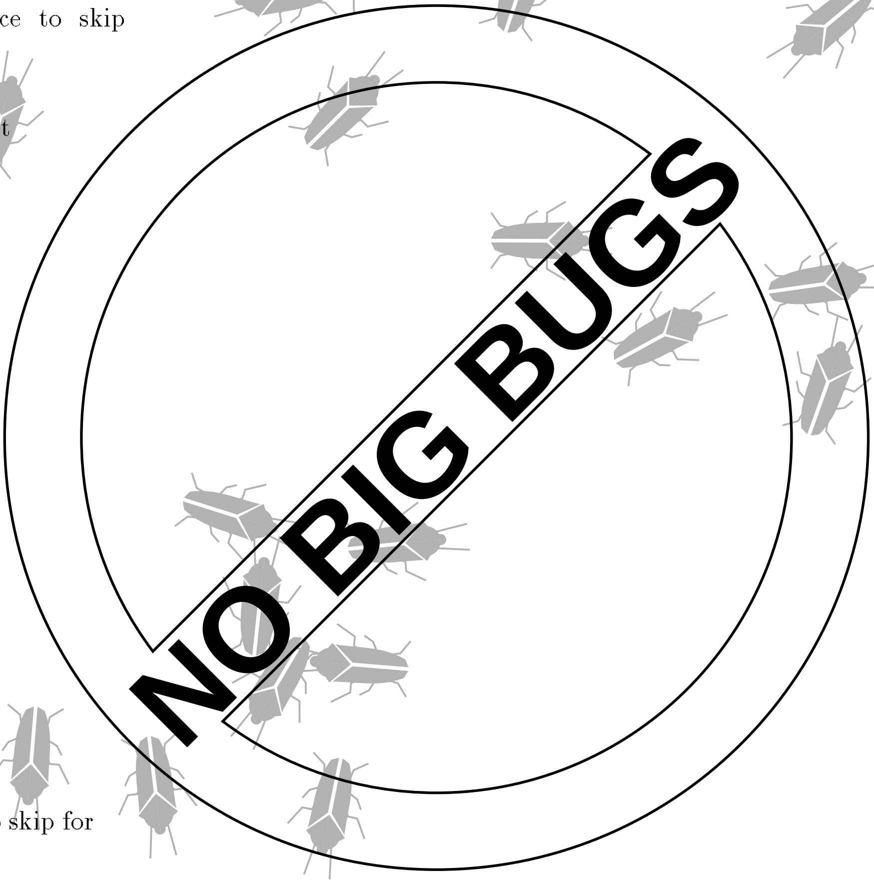
oddsidemargin.

newfontcmd The command of the "currently used font", which doesn't really have to be a font command at all.

para Whether the end of this line will mark a new paragraph.

parindent The amount of indentation for the first line of each paragraph.

parskip The amount of vertical space to skip in-between paragraphs.



rightmargin The margin at the right hand side of the page. The LaTeX words textheight and textwidth don't mean anything to LameTeX. They get translated into left, right, top, and bottom margins on the assumption that the textheight and textwidth commands apply to a normal-sized 8.5x11 inch paper.

savetype The currently saved state, saved by the save command.

topmargin The margin at the top of the page.

vspace The amount of vertical space to skip for the next paragraph.

welem The number of elements in the word list.

wlen The total length of all the words in the word list, including a single space between each word.

wlist The word list itself. Feel free to mess around with it. Please make sure to set "welem", "wlen", and "woids" to appropriate values or it'll break! The correct way to add stuff to wlist is put doing an indexed get and put into the array. The maximum length of wlist is 100 elements. It's defined at the top of format.ps if you want to change it.

woids The number of elements in the word list that are actually words.

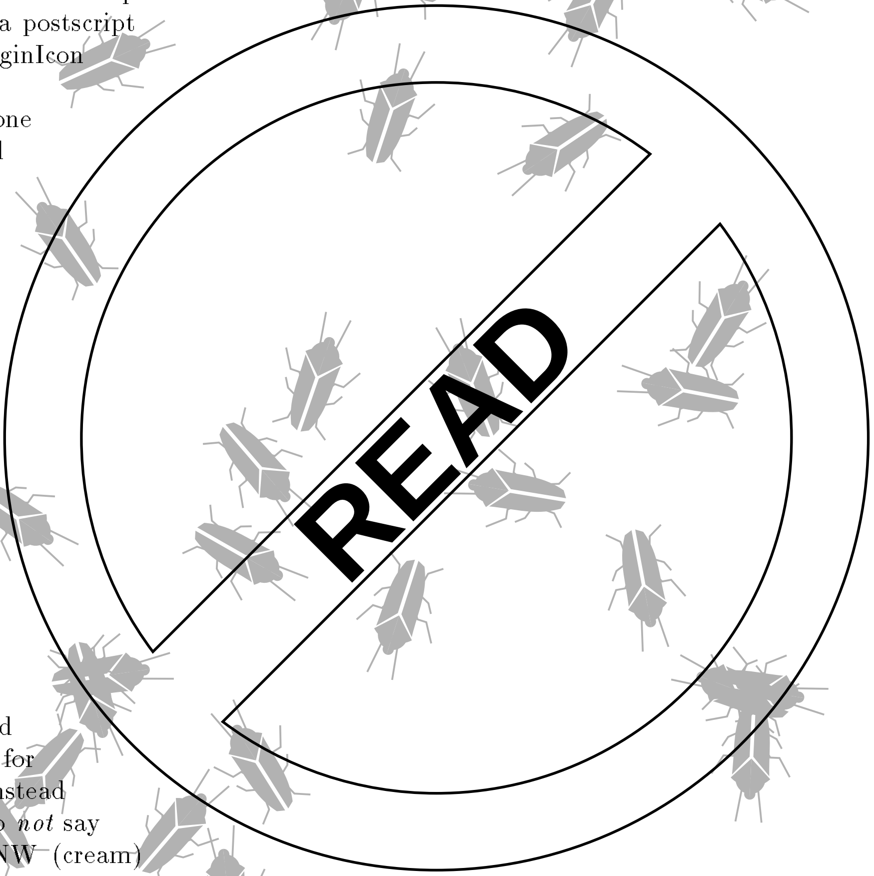
xpos The current X position.

ypos The current Y position of this line.

6.27 The Icon Hack

Notice how each page description has a different definition for LeftMarginIcon. This is because each page is a little different and "where to put icons in the left margin" varies a little.
So putting the little Wizard in the text is as simple as starting a stealth environment, starting a postscript environment, and called the /LeftMarginIcon routine to display the Wizard.

Notice that the LeftMarginIcon uses one (or two) formatdict variables: ypos (and maybe bpos). Because the LeftMarginIcon routine is being called from inside the page, there is no need to do a fancy dictionary lookupt to get their values.



6.28 Doing it Yourself in PostScript

If you want to schlep a word onto the wordlist, say "(elephantine) NW" which is the PostScript command that puts the word "elephantine" onto the word list. A space will automatically be added for you. Do *not* put a space in your words. Instead use two PostScript calls. For example, do *not* say "(ice cream) NW". Instead say "(ice) NW (cream) NW".

If you have a really long sentence, then you can just say "(ice cream) Parse". Parse is a PostScript routine that will break a string up by spacing into "(ice) NW (cream) NW".

If you want to schlep some horizontal whitespace onto the wordlist, you can use the LaTeX\hspace command. If you really want to do it in postscript, then "54.0 HSpace" is the PostScript command to add 54 points of horizontal space to the word list. There are 72 points in an inch.

Also, naturally you can gronk the value of any of the variables in the fontdict directly. Grok-p? Keep in mind that if you change them in PostScript instead of in LameTeX, then LameTeX will "lose track" of them and may not reset them for you. You will have to put them back to their original values yourself.

6.29 Using\pscmd to Do It When You Want It

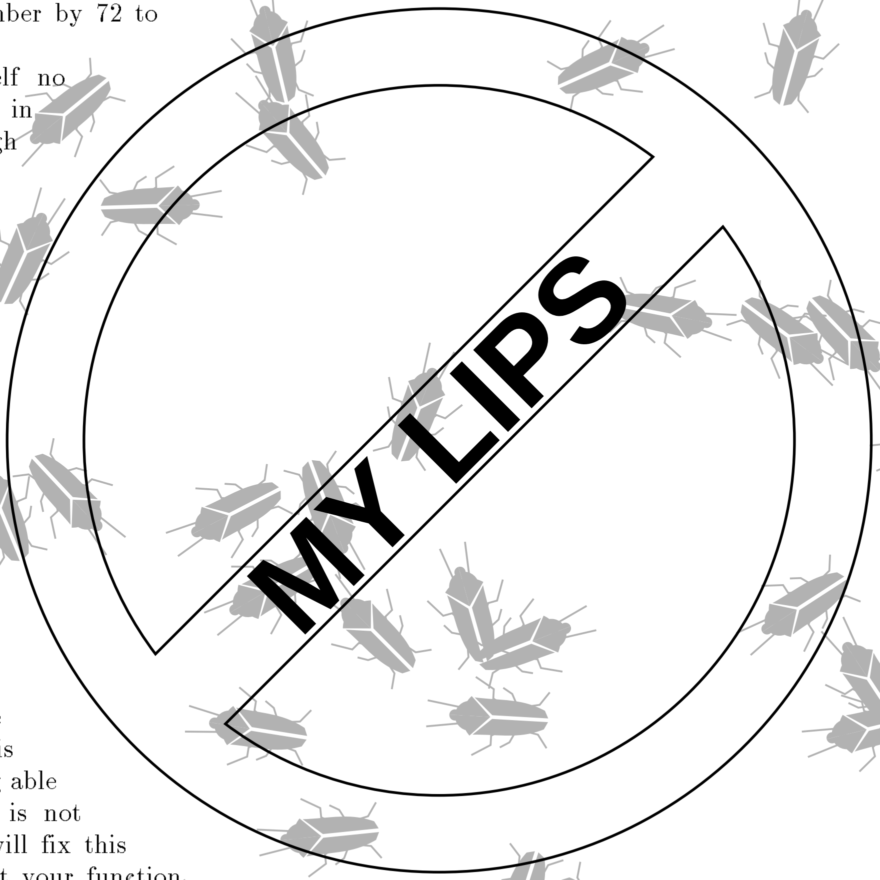
the white-painting applied to the whole line, not just parts of it. You couldn't paint just part of a line white.

Well, here's how. I'm going to tell you twice. This section is the easy slick way. The next section tells you the *same thing*, if you want to get your hands dirty, which I encourage.

Anyway, for the easy way, take a look at `example5.tex`.

Here I define a command /Smiley that alternately either draws a picture without changing the currentpoint, or calls READJUST.

Sometimes LameTeX will be building a line made out of 10 point type when suddenly the user wants to get 30 point type. Well, since the line started on this 10 point line, all the 30 point letters are too tall and will overwrite the previous line just above the current line! There is a PostScript command that LameTeX calls automatically when such oopses occur. That's /READJUST. It takes a vertical length to skip from the stack, in points. There are 72 points in an inch. I have some command "in" defined somewhere that just multiplies any number by 72 to convert it into points.



You can call /READJUST yourself no problem. For example, I do it here in example5.tex to make the line tall enough to accept the smiley face.

Don't forget! At the end of every page, everything gets wiped by the PostScript `restore` routine. Here Smiley is defined On-Page in the middle of page 1. On page 2 it would be undefined. If you want your routine to hang around for page two, you either need to define it In-Between Pages (the bad wicked way) or define it in /StartPage (the good way of lightness and being).

Notice that the\pscmd command in placed inside braces. This delimits what is now the "Smiley" environment. For example, if Smiley had changed the font, the font change would have remained in effect until the end-brace delimiter. The way this is implemented is really for fonts, and being able to add your own PostScript functions is not something I had envisioned at first. I will fix this later, but for the time being realize that your function Smiley, if it not delimited, will be executed at the beginning of each new line. Also, any font changes like italics or boldface will kill your Smiley environment. Try taking out the braces around the \pscmd and you will see what I mean.

I hope you can see how you could use this to include really big pictures, not just small smiley faces. You get to have complete control over how big the picture is, by passing its width and height to READJUST. Alternately, you can modify the wlist array directly if you want something special. In later versions, the READJUST command will be able to handle non-rectangular boundaries just like pages do.

There is no way to do this currently unless you modify the Page on the fly by defining the "current page" as something new (including the box around the Smiley" and forcing a re-assessment of the dimensions of the page.

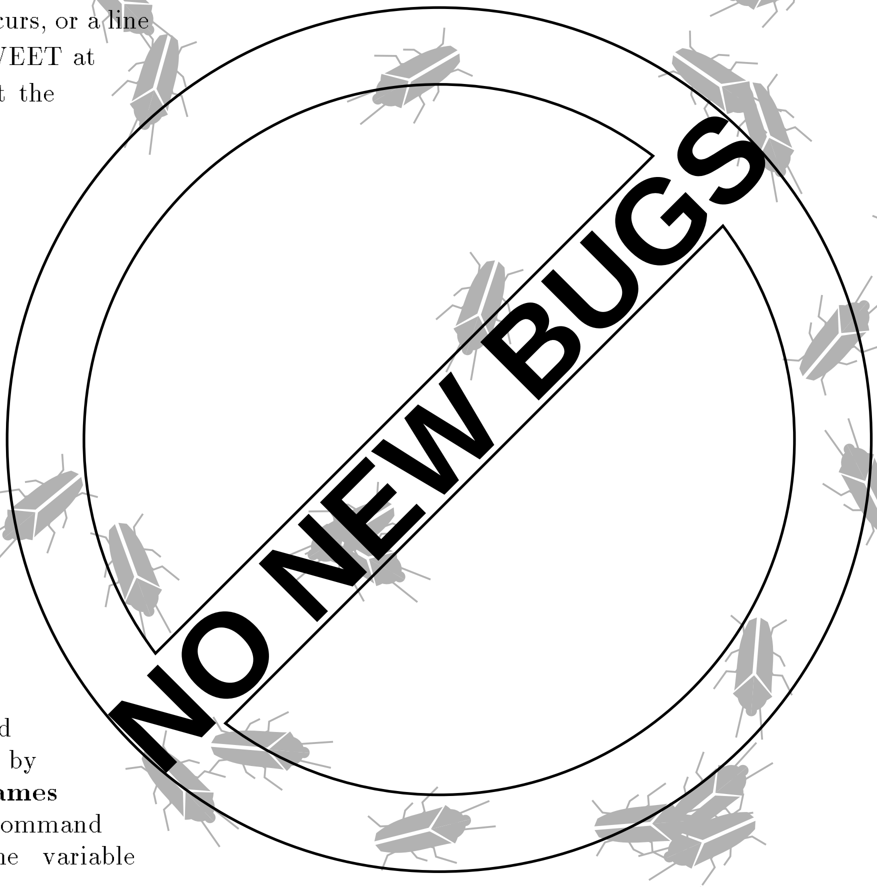
Here are the rules for a routine that you use with\pscmd:

1. When your routine is called, the currentpoint will be set to where the next word would normally be placed. Make sure that the currentpoint is defined when your routine is finished. You can accomplish this neatly by just doing a Postscript "gsave" at the beginning of your routine and a "grestore" afterwards.

2. Don't forget that you have access to the entire word list and format dictionary.

3. When your routine gets called, either false or true will be on the stack. The value will be "false" if LameTeX is building the word list and "true" if LameTeX is printing the word list. You gotta either use this or pop if off the stack by making the first command in your PostScript routine be "pop".

4. When you split up a word, like by saying `Will you be my SWEET \pscmd /drawheart HEART?` LameTeX will make an attempt to put Sweet and Heart all in the same word. It won't try really hard though. There is no concept of "glue" in LameTeX. I gotta add this sometime soon. So if a pagebreak occurs, or a line break, LameTeX will gladly put SWEET at the end of one line and HEART at the beginning of another. Oops!



6.30 Behind the Scenes of\pscmd

Remember the "Word List"? You too can schlep things onto the word list. This is exactly what the\pscmd does, but if for some reason it doesn't work or you want to try fiddling with the PostScript code directly, it's simple. Here's how.

The most interesting part is schlepping a command onto the word list. Take a look at example3.tex.

To schlep a command onto the word list, I have to first register the command by placing it onto the end of the **fontnames** array. From now on I just refer to this command by its number which I store in the variable FUNKYnum.

The actual syntax for schlepping a command onto the world list is to say " [FUNKYnum false] NewFont", where FUNKYnum is an index to my function in the **fontnames** array, and the boolean is false if I want LameTeX to add a space after this NewFont command. If the boolean is false then LameTeX puts no space between a word before the NewFont command and a word afterwards.

The line " [temp false] NewFont" is important. Once I have started playing God with the LameTeX PostScript code, I should remember to put the font back to the way it was.

I hope it's obvious that the routine would't have to be a Font-changing routine at all. It could be any routine. Here are some things to keep in mind though. Remember the flow chart from the chapter on pages? The routine will be executed *twice*, once when the word list is being compiled, and once when the word list is being printed.

6.31 The Wigglies Hack: using SHOWIT

I have written a PostScript hack called `format.ps` that does all the major formatting for LameTeX. The command that it uses to show words is the SHOWIT routine, which is defined as follows: "/SHOWIT { show } bind def".

Feel free to redefine this routine if you want to change the way the words look in any way. The routine will be called with the currentpoint set to the appropriate place and the string to be printed on the stack.

I wrote a cool hack based on this. Try it out with example4.tex. And don't forget: **When you play God with LameTeX's PostScript output, be sure to put things back!** When I have finished hacking up SHOWIT with my own definition, I put it back by saying "/SHOWIT { show } bind def".

You can't do this stuff with TeX macros! (Although you could argue I suppose that only a loony would try...)

6.32 Forcing Word Positions with NW

NW is defined as follows: "/NW NextWord bind def". NextWord is the routine that is called repeatedly, once for each word, to place the words in the word list. Feel free to modify this routine too if you want to perform some operation to the strings before they get included in the word list.

6.33 The Skew Page Hack

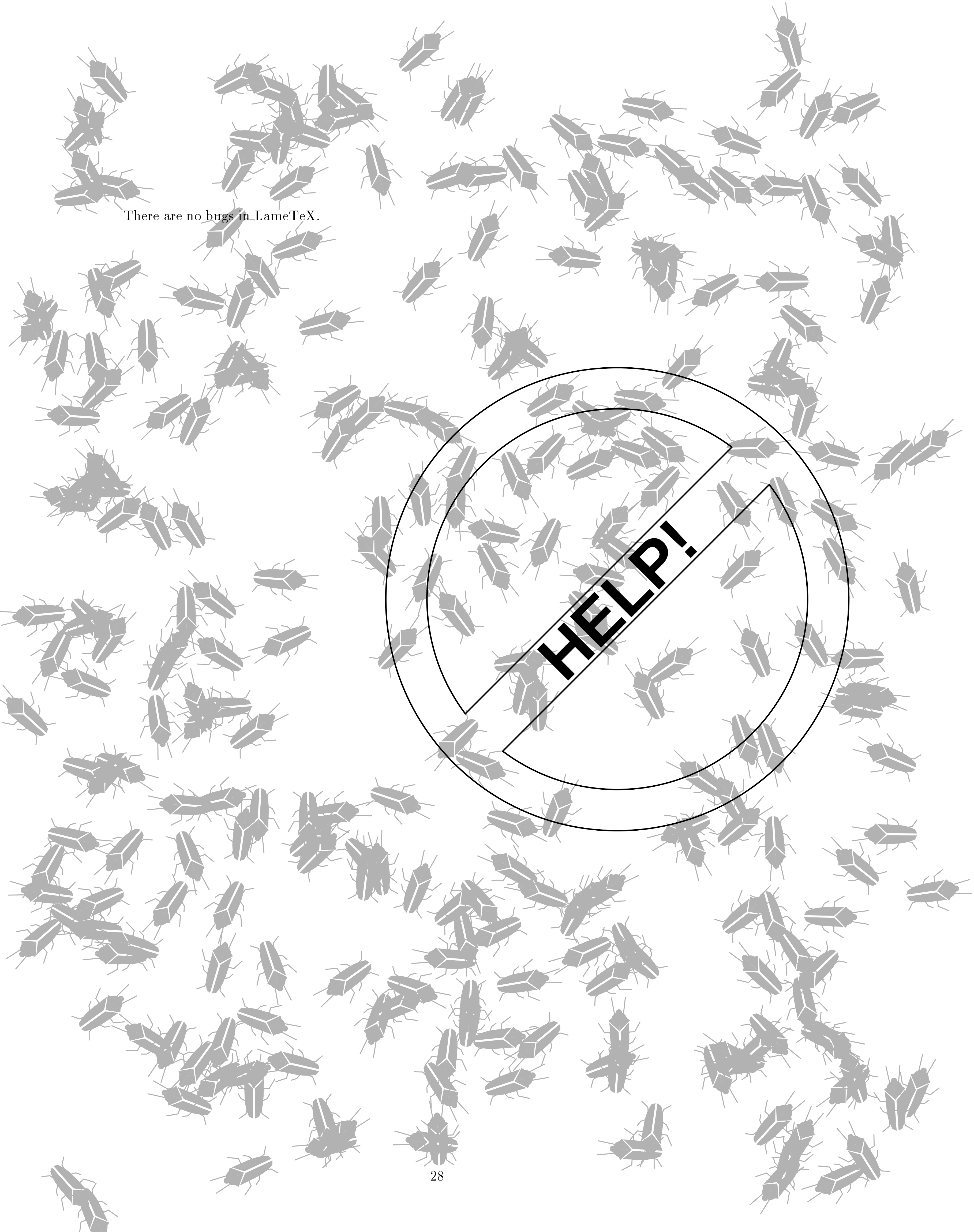
Look in the files `starting.tex` and `page_skew.ps` for an example of diagonal text. LameTeX is not really capable of placing text in any way except horizontally (unless you redefine the SHOWIT and NW routines). So therefore, in order to get diagonal text, the page must be slanted. So although it looks to the user like the text is slanted and the page is an upright rectangle, to LameTeX it appears as if the page is a very wide page in the shape of a diamond on its end. And the text appears horizontally.

6.34 The Checkerboard Hack

Check out `example6.tex`. Here is an example of two virtual pages being placed on a single physical 8.5x11 inch piece of paper, using a variable flip to determine which virtual page comes "next". I hope you can see that it wouldn't be hard to define several virtual pages on a single physical page (like for making index cards or mailing labels).

6.35 Bugs?

There are no bugs in LameTeX.



Chapter 7

The Future of LameTeX

Most of my effort was to make this understand LaTeX code. Now I can concentrate on other functionality, like making it work better, understand more LaTeX functions, and - best of all - providing easier ways to include PostScript hacks into the document.

I want to try out some experiments with on-line document viewing. There's no reason why a TeX-like language like the Stealth Commands couldn't drive PostScript animation through a previewer or let the user include audio in the document.

Although I love PostScript dearly, more coding really should go into C++. I'll let PostScript do all the fancy graphics, but C++ should do more of the decision making, because it is very difficult to write and maintain large programs in Postscript version 1.0. I would make use of some fancy PostScript interpreter to do some of the PostScript processing while processing the input file.

LameTeX should in general produce a smaller output. I would like to write a compress mode to make sleek uncommented PostScript.

There is no glue in LameTeX currently, which means that two words that happen to be side by side may have a pagebreak or a linebreak in-between them. There is no way to prevent this in the current version of LameTeX.

You can't use a previewer program that looks for the fancy Adobe PageBreak PostScript comment. Sorry. The PostScript program doesn't "know" where the page breaks are going to come until run time.

I would like to write better tools and better documentation so that people unfamiliar with PostScript can still try out a few hacks like grayscale or font changing.

Obviously, LameTeX should handle spacing better. It should do vertical alignment. LameTeX should be smart enough to handle the TeX hyphenation dictionary and be able to handle the fonts directly, which are currently being done in a big hack that barely works.

Using normal PostScript scaleable fonts should be simple and let the user save a lot of room in the PostScript output file.

The user should be able to define new LameTeX commands. This includes normal ones attach to LaTeX made with\newcommand\newenvironment\newtheorem, and special Stealth ones hooked either to C, C++, or PostScript code.

I'm hoping to hear a slew of requests for enhancements and (ahem) bug fixes! So I should have plenty to do.

I'm going to have to do some investigation to see if I can "trick" LaTeX or TeX into giving me a raw paragraph with a bounding box. If so, then I could hook up a great deal of functionality like math mode by fooling TeX into making the output and then snarfing the result.

The token parsing is done all wrong and should be cleaned up and made to model LaTeX parsing.

TeX is a trademark of the American Mathematical Society.

PostScript is a trademark of Adobe Systems Incoporated.

LameTeX ain't a trademark!

Chapter 8

Errors

If you get an error like this “stackoverflow offending command: pathforall Stack: [/coords -mark- .....] ”, this means that the path you are using for the page is very curved and not particularly flat. PostScript is having trouble keeping track of all the points! You should double or triple the setflat value.

If you get an error undefined in\end, you ended a\begin{postscript } command with something else that was not\end{postscript}.

A lot of problems come from having your margins too wide! Make sure that you margins are small enough so that your text will fit around the pictures.

If you get an error “false false false false false false false stack overflow” then you’re using\pscmd on a routine that doesn’t handle the incoming boolean. Just add a PostScript `pop` to the beginning of the PostScript routine you call with\pscmd.

Chapter 9

Commands

This is the complete list of commands that LameTeX understands. It's formatted in an easy to read alphabetically sorted list.

The stealth commands are:\stealth\postscript\includeps\pspage\ignore\pscmd\STEALTH

The normal LaTeX commands are:\#\%\&\Huge\LARGE\Large_\addtolength\backslash\begin\bf\bigskip\center\chapter\chapter*\clearpage\description\document\documentstyle\em\end\enumerate\flushleft\flushright\footnotesize\hspace\hspace*\huge\include\it\item\itemize\label\large\ldots\medskip\newlength\newline\normalsize\par\paragraph\paragraph*\part\part*\quotation\quote\raggedleft\raggedright\ref\rm\sc\scriptsize\section\section*\setlength\sf\sl\small\smallskip\subparagraph\subparagraph*\subsection\subsection*\subsubsection\subsubsection*\tiny\today\tt\verse\vspace\vspace*\

Chapter 10

Plain Text

If you want to make LameTeX produce plain text output, which is to say a plain ASCII output, it can do that.

Just type lametex -t yourfile.tex and the result will be placed in the file yourfile.txt.