

# Hints on using L<sup>A</sup>T<sub>E</sub>X to produce PDF

Sean R. Eddy

HHMI & Washington University School of Medicine

Version 1.1  
31 July 2001

The purpose of this document is to recommend how to write a L<sup>A</sup>T<sub>E</sub>X paper in such a way that it can be easily typeset in either PostScript or PDF formats, for printing or electronic publishing, respectively.

The information in this document is derived partly from relevant documentation and partly from trial and error. As neither source of information is entirely reliable, caveat emptor.

## The problem: bitmap fonts

L<sup>A</sup>T<sub>E</sub>X together with dvips produces excellent PostScript for printing. In principle you can convert a PostScript file to PDF using ps2pdf, part of the free Ghostscript package. Unfortunately this is exactly what you do *not* want to do.

The main problem is fonts. dvips, left to its own devices, will use bitmapped (“Type 3”) fonts instead of scalable Type 1 fonts. Therefore the apparently simple procedure of:

```
> latex foo.tex
> dvips -o foo.ps foo.dvi
> ps2pdf foo.ps
```

often doesn’t work too well. The file may print fine, but it will look horrible on any computer display (e.g. the Adobe Acrobat viewer). One common way around the problem is to use one of the fonts that’s built-in to PDF, such as Times or Helvetica, instead of L<sup>A</sup>T<sub>E</sub>X’s default Computer Modern fonts. However, this doesn’t fix all your problems, only most of them; and maybe you really do want Computer Modern.

To check in Acrobat if your PDF file is using any ugly Type 3 bitmap fonts, go to File > Document Info > Fonts and look in the Font Info window. What you want to see is all Type 1 scalable fonts being used.

## The solution: pdf<sub>l</sub>atex

Use pdf<sub>l</sub>atex to generate PDF directly from your L<sup>A</sup>T<sub>E</sub>X source files:

```
> pdflatex foo.tex
```

pdf<sub>l</sub>atex has some quirks with how it deals with figures and (especially) the feynmp package we use for producing Feynman diagrams. Those quirks necessitated this document.

## A suboptimal alternative: ps2pdf

You can produce reasonable PDF files from PostScript if you take a few special steps. This is not recommended, but the reasons that it fails are possibly instructive. The commands you want are these:

```
> latex foo.tex
> dvips -Ppdf -G0 -o foo.ps foo.dvi
> ps2pdf foo.ps
```

The key points are:

- The `-Ppdf` option makes `dvips` embed Type 1 fonts. This has no effect on PDF built-in fonts like Times, but is absolutely essential for the default Computer Modern fonts.
- The `-G0` is necessary though I don't understand what it does, and you may not think it's needed if you were to take a quick glance at a document prepared without it. However, look close; without it, certain combinations of letters such as the "fi" in "final" will appear as a £ sign instead.
- You *must* use a ghostscript version  $\geq 6.00$  or this will not work. Older ghostscript versions would not embed Type 1 fonts properly. Most current UNIX installations, including Linux distributions, are shipping with ghostscript 5.10 or 5.50. The current release of ghostscript is 7.00 and is freely available.
- If you're using the default L<sup>A</sup>T<sub>E</sub>X font, you have to have Type 1 fonts available for Computer Modern. Free versions of these seem to be part of the standard Linux kpathsea installation, in a font package called `bluesky`.

Although this will deal with almost everything, certain symbols apparently are still not rendered properly, such as  $\text{an} =$  (equals) sign, and that's annoying.

Nonetheless, sometimes `ps2pdf` may save you. This might happen especially with producing figures, where `pdflatex` seems particularly finicky about file formats. I had an example of a figure that was produced in L<sup>A</sup>T<sub>E</sub>X by including three PostScript `.ps` files, and I didn't have access to the original `gnuplot` scripts that made the files. No amount of fiddling (including the obvious of hacking the figures into `.eps` format, and using `epstopdf` to convert to `.pdf`) gave me something that `pdflatex` would include properly. However, I was able to use `latex; dvips; ps2pdf` to create a `figure8.pdf` with all Type 1 fonts.

## Choosing Times fonts - issues with math

By default, L<sup>A</sup>T<sub>E</sub>X uses Computer Modern fonts for both text mode and math mode.

I don't like Computer Modern; it's too thin for my taste. It also makes your document instantly recognizable as a L<sup>A</sup>T<sub>E</sub>X document. I prefer a little more mystery. You can change the text font to Times (my current favorite) simply by using `\usepackage{times}` but that only change the text, not the font used for math. Ideally you'd use the same font family for both text and math. You get this by also using `\usepackage{mathtime}`.

Unfortunately math fonts are complex, and there don't seem to be any free versions of scalable Type 1 math Times fonts. Commercial math Times fonts are available from Y&Y. If you use the `mathtime` package, and you don't have the Y&Y fonts, you're going to get bitmap fonts in your PDF, and that's bad.

A reasonable compromise can be achieved by using the `mathptmx` package instead, which uses readily available scalable Type 1 fonts. My documents (including this one) start with:

```
\usepackage{times,mathptmx}
```

An example equation:

$$M_{ij} = \sum_{x_i, x_j} f_{x_i x_j} \log_2 \frac{f_{x_i x_j}}{f_{x_i} f_{x_j}}. \quad (1)$$

The `mathptmx` package will work for almost all equations, but be warned that it apparently does not include a full set of symbol fonts. The QRNA paper, for instance, included so many crazy symbols that `mathptmx` wasn't sufficient, so I typeset text in Times and math in Computer Modern.

## Including figures

For figure inclusion, I recommend the `graphicx` package. Including the package is a little complex because we want to set it up for PDF versus `dvips` output. The basic boilerplate for including the package and then importing a figure is:

```
\ifx\pdfoutput\undefined
  \usepackage[dvips]{graphicx}
\else
```

```

\usepackage[pdftex]{graphicx}
\fi
\begin{document}
...
\includegraphics{filename}

```

Each figure should be produced as an Encapsulated Postscript file `filename.eps`, and the `epstopdf` converter should be used to produce `filename.pdf` as a duplicate in PDF format:

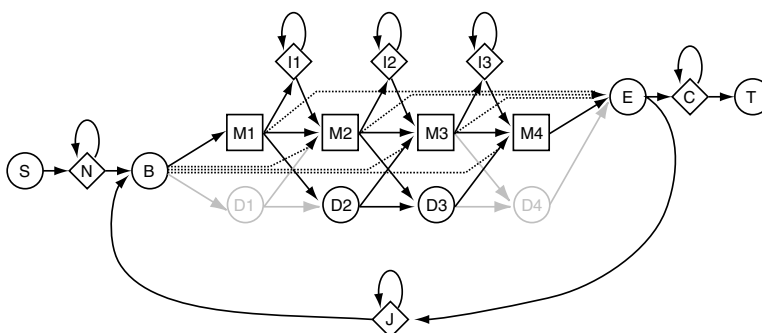
```
> epstopdf filename.eps
```

Almost any drawing program can export Encapsulated PostScript. I recommend Adobe Illustrator for most purposes.

The rationale for needing two copies of the file is as follows. `latex` can import Encapsulated Postscript (EPS) files. `pdflatex` can import PDF (.pdf), TIFF (.tif), JPEG (.jpg), PNG (.png), and MetaPost PostScript (.mps) files. How do we produce a source file that can be turned into either PostScript (with `latex; dvips`) or PDF (with `pdflatex`) if the two approaches need their figures in different formats?

The `\includegraphics` command of the `graphicx` package fortunately does not need a filename extension. It will search for an appropriate format. Therefore, if we have `figure1` available as both `figure1.eps` and `figure1.pdf`, and our source file contains `\includegraphics{figure1}`, `pdflatex` will automatically import `figure1.pdf` and `latex` will import `figure1.eps`.

An example:



The code snippet for that:

```

\begin{center}
\includegraphics[width=4in]{plan7}
\end{center}

```

`latex` doesn't care about the suffixes on your files, but it won't be able to find them unless you either use standard suffixes or give the full filename (e.g. `\includegraphics{epsfigure.1}` will work). `pdflatex`, on the other hand, rigorously enforces that your included files must have standard suffixes. This can become a problem - see the section below on Feynman diagrams.

Don't produce figures in .ps (PostScript), use .eps (Encapsulated PostScript) instead.

## Preparing figures in Illustrator

When preparing the figure, use Times font for all text labels if possible; else use another PDF built-in font, like Helvetica. When you export the file in EPS format, choose:

- no thumbnails
- no preview
- do not include fonts (this is the default)

epstopdf apparently can't convert fancy EPS files that include thumbnails and/or previews; I don't think `\includegraphics` can deal with them either.

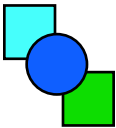
If you have trouble with `eps2pdf`, an alternate strategy is to use Illustrator's "Save As" to save twice, once in EPS and once in PDF format.

If you include fonts, that will let you use fonts in the figure other than the standard PDF fonts, and they will be "faithfully" rendered in the PDF file - except that Acrobat will struggle with the rendering, and any fonts besides the built-in PDF fonts will look ugly. PostScript printing, on the other hand, will be fine if you include fonts.

Since you're not including fonts, any font that isn't a built-in PDF font will be replaced by a typewriter font, which is probably not what you want. Use vanilla Times. (Times New Roman is not a PDF built-in.)

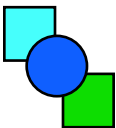
An example of various fonts in an Illustrator figure where I've included the fonts:

Helvetica  
Times New Roman  
Arial  
Courier  
Times  
Garamond



as opposed to not including the fonts:

Helvetica  
Times New Roman  
Arial  
Courier  
Times  
Garamond



Note that the built-in fonts (Helvetica, Courier, Times) render identically, whereas the others are rendered as Courier in the second case and as ugly in the first case. Not including the fonts makes the files smaller. Illustrator EPS files are huge, but the PDF files after conversion are nice and small (they're compressed).

If you just see a bunch of dots, you've got an old Adobe Acrobat reader. Upgrade your reader to at least Acrobat4.

## PDF-only or PostScript-only conditional inclusion

Sometimes there's just no getting around the fact that you want something special to happen in the PDF files that you're distributing electronically versus the PostScript files you send to the printer.

A good example is figures: you probably want to produce a single PDF file containing the entire paper, but if one or more of your figures are in JPEG, PNG, or (especially, for photographic illustrations) TIFF format, normal  $\LaTeX$  can't handle these. The strategy here is *conditional code inclusion* in your source file.

We saw conditional inclusion earlier when we included the `graphicx` package. My boilerplate code is actually a little more complex than that example, in that I define a special `\ifpdf` toggle at the same time. The code in my preambles looks like:

```
\newif\ifpdf
\ifx\pdfoutput\undefined
  \pdffalse
  \typeout{Configured for dvips (PS not PDF)}
\else
  \pdftrue
  \typeout{Configured for PDF output.}
\fi
```

```

\ifpdf
  \usepackage[pdftex]{graphicx}      % comment out if not using graphicx
\else
  \usepackage[dvips]{graphicx}      % comment out if not using graphicx
\fi

```

This is pretty general code - just comment out the `graphicx` lines if you're not using graphics. Now, we can do things like the following in our paper:

```

\ifpdf
  ... voila, a little photo in PNG:
  \begin{center}
    \includegraphics[width=0.5in]{photo}
  \end{center}
\else
  ... but we can't show you the result since we're formatting for dvips.
\fi

```

... voila, a little photo in PNG:



A reasonable strategy for papers would be to do conditional inclusion at the end of the paper. That way the figures appear as the last pages of the PDF (standard for submitted manuscripts), and you can print the individual figure files to a high-quality color printer yourself for submitting the hardcopy manuscript.

## Using the `feynmp` package for Feynman diagrams

Our descriptions of RNA structure grammars use Feynman diagrams extensively, using the `feynmp` package from Thorsten Ohl.

`feynmp` works by generating a MetaPost `.mp` file the first time you `latex` the file. Then you process the `.mp` file with `mpost` to generate a pair of files for each figure: `.1`, `.2`, etc. are MetaPost PostScript files for the diagrams, and `.t1`, `.t2`, etc. are LaTeX commands for the labels. Then in a second `latex` pass, `feynmp` detects that these files exist, and includes them. That is:

```

> latex myfile.tex      % generates feynfigs.mp and myfile.dvi
> mpost feynfigs.mp     % generates feynfigs.1, feynfigs.t1, etc.
> latex myfile.tex      % generates myfile.dvi which is complete.

```

Well. This would be fine – if `pdflatex` would recognize the `foo.1` as a file with a legitimate suffix. But it doesn't. We're stuck. Except... `pdflatex` will import files with the `.mps` suffix as MetaPost PostScript, so the problem is limited just to file naming conventions. That makes it easy to hack around.

The solution I'm recommending is two-fold. First, I hacked `feynmp.sty` itself, to create a modified version that will try to include filenames of the form "`foo.1.mps`" instead of "`foo.1`". The modified style file `feynmp-pdf.sty` is included with the package that this document came with, and the context diff is included at the end of the paper. We include it with a conditional in the preamble, as in:

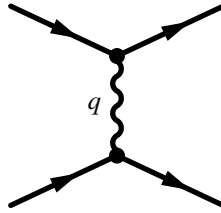
```

\ifpdf
  \usepackage{feynmp-pdf}
\else
  \usepackage{feynmp}
\fi

```

Second, we write a little Perl wrapper around `mpost` to rename the `foo.1`, `foo.2`, etc. files it generates to `foo1.mps`, `foo2.mps`, etc. This script `mpost-pdf.pl` is included with the package that this document came with, and the code is included at the end of the paper.

An example of a Feynman diagram produced by this strategy:



which was produced with this `feynmp` code:

```
\begin{fmffile}{feynfigs}
\begin{center}
\begin{fmfgraph*}(100,75) \fmfpen{thick}
  \fmfleft{i1,i2} \fmfright{o1,o2}
  \fmf{fermion}{i1,v1,o1} \fmf{fermion}{i2,v2,o2}
  \fmf{photon,label=$q$}{v1,v2} \fmfdot{v1,v2}
\end{fmfgraph*}
\end{center}
\end{fmffile}
```

and the following commands:

```
> pdflatex latex_guide.tex
> mpost-pdf.pl feynfigs.mp
> pdflatex latex_guide.tex
```

## Appendix

### Software versions

I'm on a Mandrake 7.x Intel/Linux system using the following free software:

| Program               | Package & version |
|-----------------------|-------------------|
| $\text{\LaTeX}$       | kpathsea 3.3.1    |
| <code>pdflatex</code> | kpathsea 3.3.1    |
| <code>dvips</code>    | 5.86              |
| <code>ps2pdf</code>   | ghostscript 7.0   |

All of these came standard on my Linux system with the exception of the latest Ghostscript. Ghostscript is optional; it's only needed to demonstrate getting the most out of a suboptimal solution, `ps2pdf` conversion.

```

*** feynmp.sty Sun Jul 29 17:11:09 2001
--- feynmp-pdf.sty Sun Jul 29 17:10:40 2001
*****
*** 1,3 ****
--- 1,9 ----
+ %% This file was modified from its original version.
+ %% It reads filenames of the form filename1.mps instead of
+ %% filename.1, for better pdflatex compatibility.
+ %% SRE, Sun Jul 29 17:10:03 2001
+ %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
+ %%
+ %%
+ %% This is file 'feynmp.sty',
+ %% generated with the docstrip utility.
*****
*** 155,164 ****
\fmf@graph{#1}{#2}%
\def\fmfkeep##1{\fmf@keep{#1}{#2}{##1}}%
\leavevmode
! \IfFileExists{\thefmffile.\thefmfgraph}%
! {\includegraphics{\thefmffile.\thefmfgraph}}%
! {\typeout{%
! feynmp: File \thefmffile.\thefmfgraph\space not found:^^J%
! feynmp: Process \thefmffile.mp with MetaPost and then %
! reprocess this file.}}%
\ignorespaces}
--- 161,170 ----
\fmf@graph{#1}{#2}%
\def\fmfkeep##1{\fmf@keep{#1}{#2}{##1}}%
\leavevmode
! \IfFileExists{\thefmffile\thefmfgraph.mps}%
! {\includegraphics{\thefmffile\thefmfgraph}}%
! {\typeout{%
! feynmp: Hey, File \thefmffile\thefmfgraph.mps\space not found:^^J%
! feynmp: Process \thefmffile.mp with MetaPost and then %
! reprocess this file.}}%
\ignorespaces}
*****
*** 174,183 ****
\begin{picture}(\#1,\#2)
\fmf@graph{#1}{#2}%
\def\fmfkeep##1{\fmf@keepstar{#1}{#2}{##1}}%
! \IfFileExists{\thefmffile.\thefmfgraph}%
! {\put(0,0){\includegraphics{\thefmffile.\thefmfgraph}}}%
! {\typeout{%
! feynmp: File \thefmffile.\thefmfgraph\space not found:^^J%
! feynmp: Process \thefmffile.mp with MetaPost and then %
! reprocess this file.}}%
\ignorespaces}
--- 180,189 ----
\begin{picture}(\#1,\#2)
\fmf@graph{#1}{#2}%
\def\fmfkeep##1{\fmf@keepstar{#1}{#2}{##1}}%
! \IfFileExists{\thefmffile\thefmfgraph.mps}%
! {\put(0,0){\includegraphics{\thefmffile\thefmfgraph}}}%
! {\typeout{%
! feynmp: File \thefmffile\thefmfgraph.mps\space not found:^^J%
! feynmp: Process \thefmffile.mp with MetaPost and then %
! reprocess this file.}}%
\ignorespaces}

```



```

#!/usr/local/bin/perl

# Renames MetaPost files to .mps files, so pdflatex will read them.
# Usage: mpost-pdf.pl <.mp file>
#
# Example:
#   pdflatex foo.tex           % generates .mp file and incomplete .dvi
#   mpost-pdf.pl feynfigs.mp   % generates feynfigs1.mps, feynfigs.tl, etc.
#   pdflatex foo.tex           % generates complete .dvi file

$mpfile = shift;

if ($mpfile =~ /(\S+)\.mp/) { $basename = $1; }
else { die "$mpfile doesn't end in .mp, I don't recognize it"; }

$output = `mpost $mpfile`;
print $output;

if ($output =~ /\d+ output file.? written:/) {
    $n = $1;
    for ($i = 1; $i <= $n; $i++) {
        rename "$basename.$i", "$basename$i.mps" || die "rename failed at $i";
        print "Renamed $basename.$i --> $basename$i.mps\n";
    }
}

```

Figure 2: Perl code for `mpost-pdf.pl`