Real Life LaTeX: Adventures of a TeX Consultant

Amy Hendrickson
TeXnology Inc.
57 Longwood Avenue
Brookline, MA 02146
USA
amyh@ai.mit.edu

Fortunately, the life of a LaTeX consultant can be varied and the activities diverse. In my fourteen years working with Donald Knuth's wonderful language, I've spent time teaching LaTeX, writing special-purpose macro packages for, among other things, database publishing, tables that continue for hundreds of pages, training slides, software documentation, and PDF production, in addition to my major activity — writing and supporting multiuser macro packages for publishing companies. The programming capabilities of the language are immense, and it has been fun exploring a tiny part of its possible applications.

In this paper I'd like to share some observations, especially in the areas of designing and supporting multiuser macro packages, the use of PostScript in design, and some of the capabilities of LaTeX as a generator of PDF.

Preparing and Supporting Multiuser Macro Packages

The first multiuser book macro packages that I authored were written in the early 1980s; my first journal macro package was the original version of RevTeX, a widely distributed macro set used by the American Physical Society, which I wrote in the late 1980s. Since then I've written *many* more, and am currently supporting more than thirty journal styles and four book styles that I've written for three different publishing companies. Here are some concepts I've learned in the process.

Designing the Macro Package. First of all, conceptually, there are some critical differences between preparing a macro package to be used once, and one that is to be used by many people over a series of years. Planning ahead is crucial for the multiuser package, which must be both flexible and inclusive, as well as matching the specifications of the publishing company for appearance and functionality, since it will:

- Be used by many authors on many platforms, and even in many countries.
  - Must be flexible enough to accomodate different versions of LaTeX and differing PostScript font naming conventions.
  - Must include capability for all ordinary LaTeX commands since some author will want to use one of them.
  - Must be as easy to use and document as possible.
- Must be easy as possible to change and support.

A nontrivial set of requirements!

Desirable attributes. In addition, there are other considerations which may not be ironclad requirements yet which make the macro set useful and desirable:

- Keep commands similar to the ones used by standard LaTeX. This will mean less documentation, and fewer problems for authors.
- Include commands that are not found in the general LaTeX distribution but which are generally useful, such as lettered equations, continued captions, lettered captions, and other convenient additions or alterations to the general distribution form of LaTeX.

Process rather than product. Conceptualizing the package as a process rather than a product is helpful since, in reality, the authors or publishing company will very likely want to change the style slightly or will request additional features.

To do this, we want, first and foremost, to keep the code as simple and clean as possible. Comments should be added where necessary, to help understand why a command was written in a particular way, to make it easier to make changes to it later.

Organizing the main macro set into parts according to function, and listing the various parts may take more time when writing the code but in the long run it will make it easier to find the part that needs to be changed. Examples of this are a part for theorem environments, a part for specific font calls, or a part for equations, each designated

and numbered, with a numbered list near the top of the file to make it easier to find the particular part.

Another way I have found to simplify the package and its maintainance is to have a single main macro file which will work with either LaTeX2.09 or LaTeX 2ε. This means that when a change needs to be made, it can be made to one file, which can then be copied and distributed as both `filename.sty` and `filename.cls`. The contents of each file are identical, but the filename ending will satisfy the requirements of LaTeX2.09, which is looking for a `.sty` file; and LaTeX 2ε, which is looking for a `.cls` file. Here is the switch which I build into the main macro file:

```
\newif\ifll
\expandafter\ifx\csname LaTeXe\endcsname\relax
  %  We see that LaTeXe has not been
  %  defined so LaTeX2.09 is being used
\else
  % LaTeX2e is defined, so set ll true,
  % LaTeX2e is being used.
\global\lltrue\fi
```

This means that we can test to see if the file is being used with LaTeX2.09 or LaTeX 2ε, and make definitions in those places where the conventions for the two forms of LaTeX diverge. For instance, when setting font family sizes:

```
\ifll
  %% Provide font family in LaTeX2e form:
\renewcommand{\normalsize}{%
   \@setfontsize\normalsize\@xpt\@xiipt
   \abovedisplayskip 10\p@
....
\else
  %% Provide font family in LaTeX2.09 form:
\gdef\@normalsize{%
\@setsize\normalsize{12pt}\xpt\@xpt
...
\fi
```

Another example shows how options may be used, whether the author is using LaTeX2.09 or LaTeX 2ε:

```
\ifll \let\dooptions\ProcessOptions
\else
\let\dooptions\@options\fi
\dooptions
```

There are many parts of the code where this switch is not necessary, but for those parts where it is, this branching innovation definitely makes maintaining and redistributing the macro package easier.

Making a Flexible PostScript Font File

It is a major nuisance that PostScript font names are not identical across TeX implementations. Karl Berry's naming system is helpful but, unfortunately, it isn't universally used. So, the best solution I've found is to

1. Have a separate PostScript font file that can be used for final production but doesn't need to be used by the author who is not willing to go to the trouble of customizing it. The document will then be printed in ComputerModern for the author, but translated to PostScript in the final production process.
2. For those authors willing to modify the PostScript font file, make it as easy as possible to do so.

Near the top of the PostScript font file the author will read instructions and then see the font names that need to be changed:

```
% You may need to rename these fonts to match
% the names of the .tfm files on your system.
% If you look at the directory where the .tfm
% files are stored you should be able to make
% the appropriate substitution.
% Some TeX implementations, such as TeXtures,
% will show you the available fonts when you
% click on the correct menu item.
%
% You may write in the name your system uses
% if you don't find it already written below.
%
% Change the definitions below,
% if necessary ====>

% Times-Roman
%% the Berry names:
\def\timesroman{ptmr}
\def\timesbold{ptmb}
\def\timesitalic{ptmri}
\def\timesbolditalic{ptmbi}

%% Another possibility:
%\def\timesroman{Times}
%\def\timesbold{TimesB}
%\def\timesitalic{TimesI}
%\def\timesbolditalic{TimesBI}
...
(Similar for Helvetica and Courier,
or other special font names)
...
%% <==== End of changes needed.
%% Please do not make changes below this point.
%% !!!!!!!!!
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

The authors should not have too difficult a time making this modification. We can then use the definition later in the file, after adding `\space` to the end of the font definition:

```
%% Times-Roman
\xdef\timesroman{\timesroman\space}
\xdef\timesbold{\timesbold\space}
... and similar xdef for other fonts names
```

And then we can use them for all the special use fonts that are necessary, without the author having to be at all aware of these commands:

```
\font\titlefont= \helvetica at 16pt
\font\titlethanksfont=\helvetica at 8pt
\font\cccfont=\timesroman at 7pt
\font\subtitlefont= \helvetica at 12pt
\font\specialsectionfont= \universebold at 18pt
\font\affilfont=\timesitalic at 8pt
\font\emailfont=\timesroman at 8pt
\font\communicatedfont=\timesitalic at 8pt
...
```

Macro Package Distribution

Perhaps this is obvious, but the macro packages are typically distributed from an ftp or Web site. Authors are directed to a site by their publishing company and then download the files. A `readme.txt` file can explain the function of each of the files. This system has many advantages, including the fact that the macro set can easily be changed and a new set of macros or documentation dropped into the ftp or Web site. The authors can be instructed to download the files at the time that they do their book or article so that they are sure to have the current versions.

Supporting multiuser macro packages

The complete macro package typically will include a sample file demonstrating every command that is unique to the package, and options which the user may have, as well as a template file with the commands listed in correct order so that the user may copy it and fill in the arguments to at least start his/her paper or book. The final set of files, which are very important to the success of the package, are the documentation files.

Documentation. Frankly, I don't like to read documentation, and I bet you don't either. However, we need to be able to get the information somehow, or transmit the information if we are writing a macro package.

My method, which I hope is helpful to authors, is to provide many examples of code and results: *show* rather than *tell*. I believe that this makes it easy for the author to see what command to use, by comparing their needs to the examples of typeset text, and then examining the code needed to produce that text. The author downloads the documentation file, runs LaTeX on it, and can print it on their own printer.

Another helpful technique is to provide the documentation file in PDF form. Since one of the main problems is getting people to read the documentation, having it presented in attractive PDF form with color and hypertext-linked table of contents, bookmarks, and index, helps authors get started using the package. They may view the PDF file before they have figured out what the various parts of the package are used for, and even, perhaps, before they have figured out how to run LaTeX on the `.tex` form of the documentation. The PDF file can sit on the publisher's Web site, and the author can read it with a Acrobat Reader enabled browser program.

Offering Author Support. Authors of the complex technical material that is usually typeset with LaTeX are undoubtedly very smart, but not necessarily very familiar with LaTeX. Often they are motivated to use it for their book or article, and are quite reassured to know that they can ask a question or modify the macro set to their liking. That is one reason for offering TeXnical support.

Another is that authors may be stuck on one small problem, which they can work out easily with a little help. Typical of this kind of problem is the author who can't figure out how to get the PostScript font file to work on his system. Another very common sticking place is the use of BibTeX, which is often troublesome, but yields with the help of a few suggestions.

Some authors totally refuse to read the documentation. This is aggravating to the person doing support, but the author usually can be directed politely into performing this reasonable task.

Many authors want additional capabilities. If you have not made all the normal LaTeX commands available, you will very likely hear about it, and have requests to make a command like, for instance, `\thanks{}` work in all kinds of unlikely places.

Many also want some new environment or feature. If your contact at the publishing company thinks that it is worthwhile to provide this new capability for the author, then a decision needs to be made if this would be generally useful. If so, add it to the general macro package and to the documentation; otherwise make a special version of the macro file for that particular author.

Finally, sometimes an author may discover a bug in the macros or documentation. Of course, we try our best to avoid this, but it does happen. In this case, we must change the macro file and/or documentation, and drop it back onto the ftp site so that subsequent users don't experience the same problem.

A Plea For Good Design

Many books and articles done in LaTeX use, to be charitable, a timid design. Some publishing companies distribute books done with the standard distribution LaTeX book style. Anyone who values handsome typesetting and understands the capabilities of LaTeX will find these books painful to behold, knowing that they are totally unnecessary.

No excuse for the techie look! First of all, even if the author submits his book in the default LaTeX book style, the publishing company can supply a macro file which will reinterpret the marked up commands, re-run LaTeX on the file, and, with minimum effort, produce a book with a handsome, professional appearance. Second, as far as I am aware, there are no limitations in implementing *any design* when using the combination of LaTeX and PostScript.

TeXnical Capabilities: Using PostScript with LaTeX

The possibility of combining LaTeX and PostScript code in the same macro package opens up many more options for the book designer who, without the knowledge of this potential, might be much more conservative in their design choices.

How it is done. Since we usually print books and journal articles done with LaTeX by converting them to PostScript with a driver program, we can also include raw PostScript commands in the macro file, which can then be passed, unchanged, by the driver program to the final PostScript file. As well as allowing us to add PostScript graphic effects to a macro file, there is also the capability of writing a macro which will include PostScript code which may be altered according to arguments given to the macro.

Here is a rather trivial example, but it demonstrates the principle of using LaTeX information to produce PostScript code. Once you understand that this will work you might imagine many other uses for what is essentially building PostScript code on the fly.

A LaTeX-PostScript macro can be written to position a PostScript grey or colored screen behind a particular area of text. The text is picked up as a macro argument, set in a box to be measured, and the results passed to the PostScript code, which will form a screen of the correct size, which can then be positioned underneath the given text.

First, a definition using PostScript code, designed to be used within another LaTeX macro:

```
\def\printbluescreen#1#2{%
\hbox to\hsize{\vbox to#1pt{\vss
\special{language "PS", literal
"/ChartCheckPoint save def
newpath
 0 0 moveto
 0 #1 rlineto %up
 #2 0 rlineto %over
 0 -#1 rlineto %down
closepath
0.8 0.99 0.99 setrgbcolor %% lt blue
fill
ChartCheckPoint restore
"}% end special
}}}
```

`\printbluescreen` is used in the second part of a two-part macro: the first part begins a box and the second part ends the box. This gives us a box containing the text found between the two macros which we can then measure. The results can be used as the first argument of `\printbluescreen`:

```
....
\printbluescreen{\the\boxht}{\the\pagewidth}
....
```

where `\boxht` is a manipulated version of the height of the test box, and `\pagewidth` is a manipulated version to the width of the text.[1] Each time the macro is used, a new dimension for the `\boxht` may be used, changing the PostScript commands to exactly fit the space behind the given text, in effect making PostScript code on the fly.

We can also have a normal LaTeX macro call, something like `\chapter{}`, for instance, and produce a graphic effect written in PostScript, when the macro for chapter titleblocks includes raw PostScript code that can be altered depending on the argument given to `\chapter{}`.

One of the most interesting designs I've implemented was for documentation of toolbox software packages published by The MathWorks. It had normal chapter titles but also a bar that would

[1] We need to manipulate the dimensions because the PostScript code is expecting a number and assuming that it means that number of points. Supplying a LaTeX dimension will produce a number followed by 'pt', i.e., 25.0pt, when what we need is 25.

appear in the margin, with a short version of the chapter title running sideways in a colored block, topped by the chapter number printed upright. This graphic effect would also change position depending on the chapter of the book, starting at the top of the page and gradually moving down. This striking effect was produced with a combination of LaTeX macros and PostScript code, in which information was passed from LaTeX to the PostScript code used to form the graphic.

Looking Forward: The Basic Wonderfulness of PDF

Many of you are already familiar with the Adobe Acrobat program that produces and reads PDF files. Its cross-platform and hypertext abilities, and easy user interface, make it an attractive way to distribute on-line journals and books, either on a CD or over the Web. However, we in the LaTeX world are especially fortunate when it comes to using this program because:

1. The program that produces PDF, the Acrobat Distiller, processes PostScript files. Since most LaTeX documents are translated to PostScript routinely, that means that they are ready to be distilled with no extra effort, other than being sure to use outline fonts, and a driver program run on the `.dvi` file to produce a `.ps` file.
2. The Acrobat pdfmark commands can be added to the LaTeX file, passed through the driver program unchanged, to be used by the Distiller program. This allows pre-linking of any appropriate material, as well as other features, such as generating Acrobat bookmarks automatically, changing colors of specific parts of the document, and controlling many other aspects of the final PDF file.

Examples of passing LaTeX information to PDF include the possibility of prelinking the Table of Contents, List of Tables, and List of Figures. The viewer can then click on any item in one of these environments and pop to the page listed. Similarly, cross-references, bibliography citations, and footnotes can by colored and hypertext-linked automatically, as can indices. Graphics can be added to every page, if desired, and be linked to the Table of Contents and to the Index, to make it easy for the viewer to access either of these sections of the document. Graphics can be used in specific cases and linked to the appropriate referant. For instance, a printed question might appear in the margin of a document — 'Need more information?' — and the user who clicks there would be sent to an appropriate appendix or other source of information. Another possibility for complex technical documents would be to have links to a glossary, so that the first time a term is used it would be highlighted and the user could click on it to jump to the appropriate glossary entry.

Color is free. Usually book publishers are concerned about adding color to their books because of the added cost, understandably, since each color makes the conventional printing process substantially more expensive. When using PDF, the cost of adding color is no longer an issue, so escaping from dreary black and white becomes a no-cost option. Pdfmark commands to set a particular color may also be inserted into a LaTeX macro file, so that particular parts of the document will appear in the chosen color automatically. There is tremendous potential for LaTeX/PDF book and journal production used for on-line distribution, database publishing, and many more applications.

An example of an application of LaTeX-PDF which I prepared recently was for a company that uses LaTeX for the over 400-page documentation of their statistical software package. They asked me to provide macros to produce a PDF form of their documentation for on-line help for their software.

The user of their software will now be able to click on the correct entry in the 'help' menu to access a PDF version of the documentation without leaving the original program. The Acrobat Reader program will pop up with the PDF file containing the full version of the printed documentation. When the user has found the bit of information that they need they can return to the original program which continues running in the background.

This means that instead of using only the usual RoboHelp files that must be written by the company separately from the documentation, and which would necessarily be a subset of the complete printed documentation, users will now have access to the complete documentation, with hypertext linking in the complete index, table of contents, and glossary. Graphics at the top of the page will allow users to easily jump to the contents or the index. This would seem to be a very attractive method of producing on-line help for those software companies that use LaTeX to produce their documentation.

Another innovative use of TeX and PDF can be seen in the University of Akron mathematics

professor David Story's online Calculus Tutorial, and Algebra Review. You will find them at

```
Home Page:
   http://www.math.uakron.edu/~dpstory/
e-Calculus:
   http://www.math.uakron.edu/
            ~dpstory/e-calculus.html
An Algebra Review in 10 Lessons:
   http://www.math.uakron.edu/
            ~dpstory/mpt_home.html
e-mail:
   dpstory@uakron.edu
```

I consider PDF production the cutting edge of the LaTeX world, and I look forward to exploring its potential, as I expect you will too.

Happy TeXing!

◇ Amy Hendrickson
TeXnology Inc.
57 Longwood Avenue
Brookline, MA 02146
USA
amyh@ai.mit.edu