TEX in Practice: Comments on a 4-Volume, 1400-page Series on TEX

Stephan v. Bechtolsheim
Computer Science Department, Purdue University, Computer Science Building, West Lafayette, IN 47907
Integrated Computer Software, Inc., 2119 Old Oak Drive, West Lafayette, IN 47906
`(317) 463 0162, (317) 494 7802. Internet: svb@cs.purdue.edu`

Abstract

This article discusses a four volume, 1400 page series, about TEX. It discusses two aspects: first of all I will discuss how processing a document of this size was organized. Second I will discuss extensions to TEX which I consider desirable.

A Short Introduction

In a previous draft of this article I had written about a page and a half about what pain it is to be an author, in particular the author of something as long hat and elaborate as the books under discussion; I still do not know whether there will be any real rewards since the fees paid to authors are mediocre, at best. But on the other hand: I could have quit at any time, and I decided stubbornly not to do that.

Yes, it was a frustrating activity, but now it's already the past!

Processing 1400 Pages of TEX Source Code

How did I manage a 1400 page TEX document? There are a number of additional programs that I used, which are described below.

The Computing Environment. Let me describe my environment briefly: I use a SUN 3/50 with an extra 4MB of memory (that's 8MB alltogether) and a 327MB local disk. This machine is hooked up to the department's ethernet. 3/50s are not terribly fast machines so the real processing takes place on a departmental Sequent. The source code is copied to the Sequent using the `rdist` program (remote distribution) so that only those files which changed are copied.

I use the GNU Emacs editor, as far as I am concerned, the best editor around. I have written a small TEX mode for this editor which makes the editor much easier to use with a TEX document. I cannot repeat frequently enough how important a good editor is: you spend most of the time editing your text, and therefore the best editor is just good enough. See Bechtolsheim [1988] for details.

Some Statistics. Let me give some statistics about this series of books. The source code of this series is about 70000 lines of TEX code, which occupies about 2.2 MB of storage. All `dvi` files together are about 3MB long. The size of the directory in which the processing of the book takes place, is about 16 MB, around 20 MB if all PostScript files (I have a PostScript printer attached to my workstation) are also stored. The series is subdivided into 57 different files of source code files which I call *part source files*, some of which are of auxiliary nature, but most of them are one chapter of a volume of this series. The part source files also include parts belonging to a fifth volume that is not published, but contains information such as any matters pertaining to the publisher or shell scripts which I used for a variety of functions. There are 228 macro source files which are published with this series.

For the following please note that all volumes together are regarded as one unit, and they are processed as such. Therefore, cross-references across volumes are not really any different from cross-references within the same volume.

The Input Language, the Preprocessor Used. The input language is of course largely TEX, but I made some extensions. These are *not* extensions to TEX, but codes interpreted by a preprocessor, `pretex`, which I now discuss briefly. The tasks of the preprocessor are as follows:

1. Allow for the direct inclusion of macro source code. Originally, without the preprocessor, my set up was as follows: I would store the source code of a macro which I describe in the series, with comments, in an external file. I then used `\ListVerb` to read in such an external file to generate a verbatim listing of it. In case I

wanted to use this macro source file I would use `\input` to read in the macro source file.

I ended up with tons of macro source files, as you can imagine. Note that the previously given figure of 228 macro source files does not even include the source files of all examples! The correct figure is close to 500 files.

This was the main reason for the design of this preprocessor, which allows me to do two things with lines in the part source file:

(a) Include macro source code files directly. The preprocessor writes this macro source code to external files *and* includes its verbatim listing in the output file generated by the preprocessor (this output file will later be processed by TEX).

(b) The preprocessor allows me to switch back and forth between writing a macro source file (and including its verbatim listing in the main output file) and writing comments, which appear as ordinary output in the book, but do not appear at all in the macro source file.

2. Maintain the makefile. I made extensive use of the UNIX utility `make` to process my book. The main idea behind using `make` is, of course, to let a program (rather than an unreliable human being) figure out which parts of the series need to be reprocessed after a change, and which do not.

   Administering that part is quite difficult, which was another reason for writing the preprocessor.

3. Administer overlays. As you will see later, I used an overlay `dvi` file processor (or DFP for short). Again, certain functions are controlled by input to the preprocessor.

4. Administer the inclusion of log files. There are a great many log files included with the documentation. The generation of these log files is controlled by information written to the makefile generated by the preprocessor.

There is actually another program used in building makefiles, but this is beyond the scope of this article. For `pretex` and the utility just mentioned see Bechtolsheim [1990b] for details.

Note also that before TEX is actually executed to process a part, some other TEX executions may take place, for instance, to produce log files included in parts of the series, or to generate `dvi` files of figures overlaid in this series.

Running TEX. Running TEX is the easiest part in this context. I always process only one part at a time, and if you read *TEX in Practice* (in particular volume III) then you will find that the set-up is quite similar to that of LATEX: one part of the series is processed at one time. Also, during this step, an index file is written out for each part.

Because I process only one part at a time, at the end of every processing step (`tex main`, assuming the main source file is called `main.tex`), I would rename `main.dvi` and `main.log` appropriately as, say, `intro.dvi` and `intro.log`.

After TEX. After TEX has executed, the `dvi` processor which I mentioned previously (see Bechtolsheim [1990c]), is executed twice.

The main purpose of the first execution is to extract positional information for marginal notes. I generate marginal notes using the DFP because this allows me to separate the marginal note generation completely from the generation of the text itself. I use marginal notes for the following purposes:

1. Communication with the editor (I am, of course, talking about the "person" editor rather than the "program" editor). If I have a question, I simply put this question into the margin.
2. Addition of change bars. I found change bars an extremely useful feature. In case of a change to an already edited chapter, I could mark those changed areas easily so that my editor could have yet another look at it.
3. Print index terms in the margin. To develop an index is considerably simplified, if the index information is written into the margins of the document. This way when the index is being developed it is immediately visible which index terms refer to a specific page.

The second execution of the DFP does the following:

1. Puts the date, time, version number, and file name on every page.
2. Extracts information about which fonts were used in each part and store this information. This allows the generation of a table listing all the fonts used.
3. Overlays other `dvi` files. There are a number of instances where output generated by separate TEX runs must be glued into the main document. This function is performed at this point.

Extensions of TEX

In the remainder of this article I discuss possible extensions of TEX. Theoretically, TEX can be made to do anything, but this is not really true in practice. Therefore, let me discuss what I would like to see added to TeX.

Operating system and interface related extensions. I would like to see a more flexible interface with the operating systems on which TeX runs. I am thinking of features such as opening and closing `dvi` files, asking whether or not certain `tfm` files are accessible, writing all characters to external files (including characters such as tabs and returns, I am thinking of a `\writechar` primitive analogous to `\char`). In TeX 3.0 the current line number is accessible, which is something I would have listed here if that were not the case. I would also like to be able to invoke other other programs, and I realize that TeX source code using this feature would be restricted in their portability.

Graphics extensions. I have *no desire* for any graphics extensions. The inclusion of PostScript generated figures works just fine.

Insertions and output routines. Because of the size of LaTeX's output routine and the fact that insertions are *not* used for figures and tables (only for footnotes), I would like to see an extension to TeX's insertion mechanism. It should be made more powerful to allow one to specify, for instance, the number of insertions that can be permitted on one page: both a maximum and a threshold vertical length which, if exceeded by insertion material, will prevent other material from being printed on that page. This is a very short description of what I have in mind; LaTeX has a whole set of style-file related parameters, which really should be built-in parameters of the insertion mechanism of TeX.

What I envision is a set-up in which the insertion queues are accessible to the user, so that the user can write TeX code which inquires about the number of elements in an insertion queue, the length of individual insertion elements, and so forth.

Also, when TeX's page-breaking algorithm completes a page and puts it in box register 255, the glue and penalty information around that breakpoint is essentially lost (with the exception of the setting of `\outputpenalty`). It is thus impossible, from within the output routine, to restore an old page in its entirety.

Paragraph computations of TeX. Typesetting specifications by publishers may prohibit a page break just following a heading or in the following two or three lines of text. Therefore an `\afterclubpenalty` should be introduced, and maybe one should generalize this penalty business even further.

The `\everypar` register is evaluated after the `\parskip` glue has been sent to the vertical list with the current page or vbox. This makes it difficult to use `\everypar`. Therefore, I would like to have a built-in token register `\everyvpar` which is evaluated as soon as TeX decides it is time to begin a paragraph but before TeX gets around to doing anything about it.

There should be a `\parskippenalty` as well as a `\baselineskippenalty` and a `\lineskip-penalty`.

Expansion, grouping. I would like to have access to the current level of grouping in the form of a read-only counter register. This would allow me to determine at the time a heading is encountered whether all preceding groups have been terminated (that is, I would like to be able to set up TeX in such a way that groups cannot extend beyond certain subdivisions of a document).

A boolean data type and boolean operations (`\not`, `\and`, `\or`, and so forth) should be added. It should be possible to write "real conditionals".

Doing any type of arithmetic in TeX is a bit of a pain, so I would like to see something which would allow me to write, for instance:

```
\dimen0 = 1.3 * \baselineskip - \wd0
```

Relational operators $\neq$, $\geq$, $\leq$ should be made available for register arithmetic.

Box computations. It should be possible to access the badness of a box stored in a box register. Furthermore, it should be possible *to access and manipulate each element* of the horizontal or vertical *list* of a box on an individual basis. In other words, I would like to see a generalization of primitives like `\lastskip` and `\lastpenalty`. For instance, if `\lastpenalty` is zero, then this means either that the last item was a penalty of zero, or was not a penalty. I would like to see, therefore, a reliable way to learn what each item is, not just the last one.

In particular, I would add primitives which allow access to the dimensions of the lists and list elements of boxes. One reason the insertion of change bars with a `dvi` file processor is so easy (see Bechtolsheim [1990c]), but so difficult in TeX, is that there is no way to access this information.

I personally would remove the restriction which permits `\vcenter` to be used in math mode only.

Math mode. Believe it or not: I found someone who wanted more than three different fonts per font family in math mode. I am not sure I concur with this.

Concluding remark

TEX is a *great product.* It is so wonderful, powerful, and flexible, it's worth all the effort required to learn it.

Bibliography

Bechtolsheim, Stephan v. "Using the Emacs Editor to Safely Edit TEX Sources", *TEXniques* 7, pages 195 - 202, 1988.

Bechtolsheim, Stephan v. *TEX in Practice.* New York: Springer, 1990a

Bechtolsheim, Stephan v. *A TEX Preprocessor and a* `make` *related Utility.* West Lafayette: Integrated Computer Software, Inc., Report 90-1, 1990b

Bechtolsheim, Stephan v. *A* `dvi` *File Processor.* West Lafayette: Integrated Computer Software, Inc., Report 90-2, 1990c

Knuth, Donald E. *The TEXbook.* Reading, Mass.: Addison-Wesley, 1984.