

**1991 Annual Meeting Proceedings, Part 1**

TEX Users Group  
Twelfth Annual Meeting  
Dedham, Massachusetts, July 15-18, 1991

# **TUGBOAT**

COMMUNICATIONS OF THE TEX USERS GROUP  
TUGBOAT EDITOR — BARBARA BEETON  
PROCEEDINGS EDITOR — HOPE HAMILTON

VOLUME 12, NUMBER 3 • DECEMBER 1991  
PROVIDENCE • RHODE ISLAND • U.S.A.

Inc., Northlake Software, Northeastern University/Dedham Campus, Micro Programs, Inc., Kinch Computer Company, K-Talk Communications, the Hilton at Dedham Place, ETP Services Co., Blue Sky Research, ArborText, Inc., the American Mathematical Society, and Addison-Wesley Publishing Co. for their sponsorship of various activities.

The Program Coordinator, Christina Thiele, and the Program Committee, consisting of Ron Whitney, Mimi Lafrenz, Don Hosek, and Michael Doob, deserve a big vote of thanks for their outstanding job in bringing it all together. And of course, we thank the panelists, speakers, and authors of papers in these Proceedings for their welcome contributions.

## New publications

A few weeks before the meeting, the first prototype issue of *TEX and TUG News* arrived in our mailboxes. *TUGboat* volume 12 number 2 was available at the meeting, together with a new publication, the TUG Resource Directory. Response to both has been very favorable. Details for the preparation of future issues of *TEX and TUG News* are still being worked out; volunteers are hereby solicited.

Springer-Verlag had a few copies of the new book *A Beginner's Book of TEX* by Raymond Seroul and Silvio Levy [7]. This is an English translation, and enhancement, of the former's excellent *Le Petit Livre de TEX* published in 1989. I'm pleased to see an English version of this book, and I certainly enjoyed reading it from cover to cover.

A new book on L<sup>A</sup>T<sub>E</sub>X by Jane Hahn [4] has just been published by Personal T<sub>E</sub>X. Michael Urban's *TEXniques* publication [8] is now available from GUTenberg in a French translation, *Premiers Pas en L<sup>A</sup>T<sub>E</sub>X*. Michael Doob's *A Gentle Introduction to TEX* has been republished in Czech.

## TUG Board matters

The TUG Board spent two and a half long days in meetings just before the conference, working on the many changes that have been instituted in the last year, the most important of which is probably the new election procedures. An elected Board will take office on January 1, 1992.

The Board felt that because of the significant change in election procedures, it was imperative that a new President be in place when my term of office as current President expires at the end of 1991. With a new Board being elected, and possibly not meeting until spring or summer of 1992, this would be difficult to ensure with a Presidential election by

mail ballot this year. The Board therefore appointed Malcolm Clark to serve as TUG President for the year 1992. When ballots are sent out in the summer of 1992, the office of President will again be up for election, and a two-year President will be chosen by the membership. Board members will also serve two-year terms, with half being elected every other year.

The Board also chose to fill the vacancies in the positions of Vice-President, Secretary, and Treasurer created by the resignations of those officers (see *TEX and TUG News*). Christina Thiele now occupies the chairs of Vice-President and Secretary, and Allen Dyer fills the important position of Treasurer. These officers will serve until the new Board convenes in 1992 and appoints three of its members to these positions.

It is disappointing to report that the TUG financial situation for 1991 again appears to be heading for a significant deficit. The Board is considering further cost-reduction measures at this time and we expect to place TUG on a sound financial footing for 1992.

## References

- [1] Nelson H. F. Beebe. President's introduction. *TUGboat*, 11(3):335–336, September 1990.
- [2] Kenneth P. Brooks. Lilac: A two-view document editor. *Computer*, 24(6):7–19, June 1991.
- [3] Martin Bryan. *SGML—An Author's Guide to the Standard Generalized Markup Language*. Addison-Wesley, Reading, MA, USA, 1988. ISBN 0-201-17535-5.
- [4] Jane Hahn. *L<sup>A</sup>T<sub>E</sub>X for Everyone*. Personal T<sub>E</sub>X, Inc., 12 Madrona Avenue, Mill Valley, CA 94941, USA, 1991.
- [5] Yannis Haralambous. Typesetting Old German: Fraktur, Schwabacher, Gotisch and Initials. *TUGboat*, 12(1):129–138, March 1991.
- [6] Raymond Seroul. *Le petit Livre de TEX*. InterEditions, 1989. ISBN 2-7296-0233-X.
- [7] Raymond Seroul and Silvio Levy. *A Beginner's Book of TEX*. Springer-Verlag, 1991. ISBN 0-387-97562-4, 3-540-7562-4. This is a translation and adaption by Silvio Levy of [6].
- [8] Michael Urban. An introduction to L<sup>A</sup>T<sub>E</sub>X. *TEXniques, Publications for the T<sub>E</sub>X community*, (9):iii, 1–56, 1990.
- [9] Eric van Herwijnen. *Practical SGML*. Kluwer Academic Publishers Group, Norwell, MA, USA, 1990. ISBN 0-7923-0635-X.

# KEYNOTE ADDRESS:

## Two Sides of the Fence

Nico Popelier  
Elsevier Science Publishers  
Academic Publishing Division, R&D Department  
Sara Burgerhartstraat 25  
1055 KV Amsterdam  
the Netherlands  
Internet: [n.popelier@elsevier.nl](mailto:n.popelier@elsevier.nl)

### Abstract

The purpose of this talk is to give an overview of the four days of the twelfth annual TUG meeting; it is an attempt to show that the different streams in the programme of the meeting are connected, that they are part of a whole.

Also, I make some comments and observations regarding the current status and the future of  $\text{\TeX}$ , and the future of publishing in general.

### Introduction

In his book *Zen Buddhism* [5], Christmas Humphreys writes:

How then, does it work, this faculty of the mind [the intellect] which men so highly prize and far too lightly claim to be infallible? The answer is, by the interaction of the opposites.

The purpose of this talk is to give an overview of the four days of this conference, and I will use pairs of opposites to guide me through it.

If you talk about pairs of opposites, you also talk, implicitly, about a fence, a boundary between the two opposites. And if you consider any of these fences you can ask yourself: do we make an opening in the fence, i.e. make a pragmatic decision in order to bridge the gap, to integrate seemingly irreconcilable views? Or will we remain passive, will we stay 'sitting on the fence', i.e. not decide anything? There is of course a third possibility, namely that the fence is there for a real purpose.

I hope that this conference will result in gates through the various fences I will discuss.

### Dichotomies

The first pair of opposites came into my mind very quickly: the  $\text{\TeX}$ -using author vs. the  $\text{\TeX}$ -accepting publisher. From the  $\text{\TeX}$  files we've received so far at Elsevier Science Publishers I've gotten the impression that the average  $\text{\TeX}$ -using author wants as much freedom as possible to typeset the text, the tables, the math and the figures.

He/she wants to use  $\text{\TeX}$  in any possible imaginable way and, according to  $\text{\TeX}$  experts at a few physics institutes, spends sometimes up to 50% of the total time for the article or book on its presentation.

Suppose he has to deal with publisher X, who has a  $\text{\TeX}$  macro package plus instructions to authors. Then maybe the author isn't very happy with it, since it limits him in his creativity and furthermore, since he has to deal with many publishers, he has to figure out a way of dealing with these different macro packages and instructions. A very likely solution is that he just ignores them all!

The publisher who accepts  $\text{\TeX}$  has a slightly different point of view. Of course, on the one hand, a publisher wants to be as friendly as possible to an author and accept his compuscript. But, on the other hand, a publisher wants to convert the  $\text{\TeX}$  compuscript into a printed book or journal paper in the shortest time possible with a minimal amount of effort.

There are several constraints to be met in this publication process: the house-style for the particular journal or book series, the quality of the publication (language, layout), the time it takes to publish the article or book, and the cost of all this. Most publishers are commercial firms, not philanthropic institutions, so cost efficiency is an important criterion. In most cases, the publisher would really like to see authors following the instructions.

How do you solve this dilemma? A compromise might be to agree upon a certain standard or set of

standards between various publishers. In our company, we think that we will not be able to handle  $\text{\TeX}$  compuscripts efficiently if we accept all varieties of  $\text{\TeX}$ , especially because the material ranges from very simple to very complex with lots of math and tables. Efficiency is particularly important for journals, where you have a steady flow of material, a fixed house-style and a routine way of working.

Our choice is: one variety of  $\text{\TeX}$ , namely  $\text{\LaTeX}$ . For book and proceedings projects this preference is somewhat less strong, although making a book ready for publication, in a house-style or in the style of a particular book series, complete with a table of contents and an index, is easier if the book was prepared with  $\text{\LaTeX}$  — and the author has used  $\text{\LaTeX}$  well! — than if it was prepared with plain  $\text{\TeX}$ .

Besides the problems just mentioned, there are several other matters you have to solve anyway, regardless of whether you use plain  $\text{\TeX}$ ,  $\text{\LaTeX}$  or, say,  $\text{\Psi\Y\S\TeX}$ :

- complex tables
- page layout
- font selection (other fonts than Computer Modern)
- illustrations in PostScript or other format

So now I've come to my second pair of opposites, one that will be addressed by several speakers this week:  $\text{\TeX}$  versus  $\text{\LaTeX}$ .

The key concept of  $\text{\LaTeX}$  is, as you of course know, the concept of logical design: an author writes his text in terms of abstract building blocks, in terms of the logical structure of the text. Content and layout are decoupled as much as possible. The visual structure is derived from the logical structure, and is specified in the document style.

As I said earlier, some authors appear to spend large amounts of time on the presentation of a paper that is submitted for publication in a journal: they write sets of macros ranging in size from one screen to many hundreds of lines, use any font they can find in all sorts of combinations, etcetera. This strikes me as odd for two reasons: (i) an author's main concern should be the *contents* of the article or book, and (ii) the presentation the author chooses will almost always be changed by the publisher anyway, whether he submits the material on paper, on a diskette or via electronic mail.

We have found that the  $\text{\LaTeX}$ -way-of-working is fine for both journals and books: document styles have been written for about ten journals and several books. The difference between conventionally typeset material and material produced from author-

prepared  $\text{\TeX}$  files can only be seen by a well-trained eye.

Now of course, there is much more to this type of electronic publishing than just changing the document style: a technical editor has to look at spelling, punctuation, language in general, notation, the appearance of mathematical formulas in text and in displays, the layout of tables, the page layout, spacing, hyphenation, ... a lot of work, often difficult work. The combination of usual copy-editing with  $\text{\TeX}$  requires skilled technical editors and a certain routine way of handling  $\text{\TeX}$ .

But  $\text{\TeX}$  is not the only document preparation publishers have to deal with. And so now I come to my next pair of opposites:  $\text{\TeX}$  vs. non- $\text{\TeX}$ , or  $\text{\TeX}$  versus the rest of the desktop-publishing world.

If we asked scientists who publish in one of our more than 600 journals whether they use a computer to write their articles and if so, what word processor they use, we would find enormous variety in their answers. In physics and mathematics,  $\text{\TeX}$  is used by the majority of authors, but even there you find a significant number of authors who use *troff/eqn*, ChiWriter, Word or various Macintosh word processing programs.

In other scientific disciplines,  $\text{\TeX}$  is used by only a few people—if at all! What I personally find most interesting is the many ways  $\text{\TeX}$  is used, not by mathematicians and physicists, but by people working in, say, linguistics, humanities. My next pair of opposites.

Often there is no alternative but  $\text{\TeX}$  for producing texts in languages that use non-Latin alphabets or the Latin alphabet with diacritical marks. With  $\text{\TeX}$  you can produce remarkable, often beautiful results, after you have solved dozens of problems that others, who use  $\text{\TeX}$  for texts written in English, with a lot of math and tables, have never thought of. I am fascinated by the work on

- hyphenation of other languages than English
- right-to-left text with  $\text{\TeX}$ : Hebrew and Arabic
- diacritical marks and other embellishments: Hebrew, Vietnamese
- wonderful fonts: Greek, Hebrew, Arabic, Old German, Ethiopic, Korean *hangul*, Japanese *kana*, Chinese *kanji* or *hanji*, and the many languages of the Indian sub-continent
- vertical typesetting: Japanese and Chinese

and I hope to see a lot of these types of  $\text{\TeX}$  applications during this conference. I think that, in principle,  $\text{\TeX}$  has great potential as a text composition system for authors in *all* scientific disciplines

and in *all* languages. But, I said ‘in principle’ — I will come back to that later.

Coming back to the observation that *T<sub>E</sub>X* is not the only software: when a publisher sees that he also receives papers prepared in Word and ChiWriter, what does he do with them? Does he handle them in the old-fashioned way, that is re-type the whole thing and introduce lots of typos, so that the author has to read the stuff for the umpteenth time? Or should the publisher convert it to one of the professional typesetting systems he uses? Or convert it to *T<sub>E</sub>X*, since there are several of these conversions available: WordPerfect to *T<sub>E</sub>X*, ChiWriter to *T<sub>E</sub>X*, ...

I think conversion will become important or is already becoming more and more important. Conversion of information from one format into another, from an author’s word processor X to a publisher’s typesetting system Y. Now suppose authors use *M* different word processors and that publishers uses *N* different typesetting systems: does this mean we have to wait for the development of  $M \cdot N$  different conversions? This does not appear to be a feasible solution. Conversion, or translation, via an intermediate language, a standard exchange language for text, would require only  $M+N$  different conversions, much less!

As most of you know, such an intermediate language already exists: SGML [6, 2, 4]. Aha, you might think: the fourth pair of opposites. Well, yes and no. Yes, in the sense that many people think that *T<sub>E</sub>X* and SGML are two alternatives for one and the same purpose. No, in the sense that I do not agree with this: I do not believe that SGML and *T<sub>E</sub>X* form a pair of opposites and I would like to explain why I think this is the case.

SGML is not a typesetting language, but an abstract language, or more precise: a meta-language. Just as you can define the computer programming languages Pascal and Modula-2 in BNF (Backus-Naur form), another example of a meta-language, you can define typesetting languages in SGML.

In SGML, there exists something that is called the *document type definition*. A document type definition (DTD) is a description of a class of documents. You describe a document instance, a document that is representative for a certain class of documents, say *book*, as a hierarchy of building blocks. To give an example:

```
book   = front_matter body back_matter
body   = chapter+
chapter = chapter_heading, paragraph?, section*
...
```

all the way down to the basic building blocks: paragraphs of text, mathematical formulas, ... This defines the contents of the book in terms of logical entities: you might call it ‘object-oriented writing of a document’.

An alternative is to describe the visual structure of a document, which can also be regarded as a hierarchy of building blocks.

```
book      = pages+
page     = header_block text_block footer_block
text-block = ...
...
```

These are sketches of two DTDs. A DTD defines a set of tags, you could say typesetting instructions, and their hierarchy. The set of typesetting instructions is in fact a typesetting language. So in fact I’ve just given two typesetting languages. You could also define the syntax of a language like *T<sub>E</sub>X* in SGML. Mostly however, document type definitions are written with the logical structure of a class of documents in mind.

By the way: two parallel views of one piece of text — view 1: logical structure, view 2: visual structure — can be important or even essential in pre-existing text, something that is pointed out in the draft report of the Text Encoding Initiative [10], on which Michael Sperberg-McQueen will speak [9]. For example: inscriptions found on historical sites or texts in *real* manuscripts — you know: hand-written books.

At present however, publishers do not receive a great quantity of SGML-coded material — not yet! There are not many SGML editors available and the ones that are available are not or hardly ever used by the authors one finds in normal textbook or journal publishing. Furthermore, the word processors these authors use do not have an SGML export facility. So if a publisher wants to have material available in some form of SGML, it means converting it from whatever form the material is in when he receives it — at least for many years to come.

Encoding a piece of text with SGML means

- separating form from content, presentation from function
- adding structure to a text, enriching the text

In particular, the last activity is a time-consuming one, both for the author and the publisher, but it significantly increases the potential usefulness of the information. If a text is fully tagged, as it is called in SGML, if pieces of text are identified by their function, all sorts of information can be extracted, stored and re-used. For example: the article opening and the lists of literature references.

If you use the text as part of a hypertext, links to figures, tables, references, footnotes and other parts of the text can be derived *automatically*.

But I would like to stress that SGML has nothing to do with getting a piece of text on paper or on screen. For that, you always need a separate program. So, 'SGML or **T<sub>E</sub>X**' is not a question at all, since you can't compare SGML and **T<sub>E</sub>X**. Valid questions to be asked are:

- do you combine SGML and **T<sub>E</sub>X**, SGML and Ventura, or SGML and you-name-it?
- *how* do you combine, let's say, SGML and **T<sub>E</sub>X**?

Suppose you use **T<sub>E</sub>X** as a back-end to a document-preparation system based upon SGML. What sort of problems do you encounter then? If you make a list of these problems and add ideas from various other **T<sub>E</sub>X** experts, you get a very long wish list indeed. What extensions do we need to add to **T<sub>E</sub>X**? Are we going to change **T<sub>E</sub>X** or are we going to build a completely new program?

## Future of **T<sub>E</sub>X**

I'd like to spend a few minutes of my talk on this subject, since I'm not really happy with the current status of **T<sub>E</sub>X**. If you think the following is a bit provocative, well . . . , maybe it's intended that way.

To put it simply: I think the program should never have been frozen. Its author should either have continued developing **T<sub>E</sub>X** or handed over this work to a new implementor, or preferably a group of implementors. If this happens with professional—or, if you like, commercial—software, if you do not listen to the users of your program, or if you freeze a program, the software will be as good as obsolete after a few years.

I will not try to improve upon Frank Mittelbach's excellent paper 'E-**T<sub>E</sub>X**: Guidelines for Future **T<sub>E</sub>X** Extensions' [8], which he presented at last year's meeting in Texas. Rather, I will add a few of my own comments, or observations.

A big deficiency in **T<sub>E</sub>X** is the page-breaking algorithm and the tools **T<sub>E</sub>X** offers to program complicated page layouts, for example two-column or three-column with footnotes and floating bodies of 1 or more columns. If you use **T<sub>E</sub>X** as it now is as the back-end to an SGML-based system, page layout cannot be achieved fully automatically: manual work is still required. And even though **T<sub>E</sub>X** is intended to be used by a typist, not as a fully automatic back-end system, the more work the computer does without human intervention, the better. This makes the SGML-**T<sub>E</sub>X** combination far from ideal.

The same problem occurs if you use **L<sub>A</sub>T<sub>E</sub>X**, which has a pretty complex output routine for scientific journals with a two-column layout, with lots of figures, tables and footnotes.

**T<sub>E</sub>X** users who have tried it know how difficult it is to let **T<sub>E</sub>X** typeset text—let's assume ordinary left-to-right text—in a language with lots of accented letters, ligatures and complicated hyphenation. Why are there no under-accents, multiple accents? Why is hyphenation of accented words or compound words with hyphehs such a problem? I will use a few technical phrases from my own background, nuclear physics, as examples to show that the problem of hyphenating compound words, for example, is not just a problem of, say, the German or Dutch language.

Compound words are quite frequent in Dutch, for example:

**schillenmodel-berekening**

(shell-model calculation). Most **T<sub>E</sub>X** users would like to see **T<sub>E</sub>X** hyphenate this as 'schil-len-model-bere-ke-ning', which **T<sub>E</sub>X** of course doesn't do.

But compound words of this type also occur in English:

**formation of a compound nucleus**

is hyphenated by **T<sub>E</sub>X** as 'for-ma-tion of a com-pound nu-cleus', whereas

**compound-nucleus formation**

is hyphenated by **T<sub>E</sub>X** as 'compound-nucleus for-ma-tion', instead of 'com-pound-nu-cleus for-ma-tion'.

There should have been a switch for this in **T<sub>E</sub>X**, but there isn't! Why wasn't the functionality of **T<sub>E</sub>X-X<sub>E</sub>T** and everything else I've mentioned added to **T<sub>E</sub>X** 3?

It is my opinion that **T<sub>E</sub>X** would have been a better program if its creator had agreed to re-think certain choices he had made years ago, especially when users argued their case by showing what sorts of problems **T<sub>E</sub>X** poses, as was done by several of them in articles in **TUGboat**. Barbara Beeton explained to me some time ago that the decisions regarding **T<sub>E</sub>X**'s accent mechanism—\accent or ligature, single or multiple accents, only above or also below and to the side?—were Don Knuth's decisions and his only; they were not based on discussions with other experts, which I think is unfortunate. I sometimes think—and this is not intended as a bad joke!—that certain parts of **T<sub>E</sub>X** would have looked different if Knuth had been German or Greek, because English is such an easy language to typeset, relative speaking!

And while **T<sub>E</sub>X** is superior in mathematical typesetting, there is still a lot to criticize in that

area as well. An example is the spacing between the eight basic types of math atoms, which is hardwired into the program as a sort of matrix, instead of being accessible via parameters. This results in a lot of handwork if a particular house style deviates from *TEX*'s rules. Again, I would like to refer to Frank Mittelbach's article and the work on *AMS-TEX* by Michael Spivak.

Another example: where's the missing lowercase Greek?

	upright form	slanted form
lowercase letter	?	$\pi$
uppercase letter	$\Pi$	$\Pi$

In other words: why was it arbitrarily decided that there was no need for upright Greek lowercase letters in the Computer Modern fonts?

A lot of work still needs to be done. Whoever is going to do it, I think that the successor to *TEX* 3—the matter of the name, *TEX* 4 or E-*TEX* 1 or God-knows-what, is unimportant, the important thing is that there should be *one* successor, not several incompatible systems based on or derived from *TEX*—should not be developed and maintained by

- one single person
- one or more persons all working in one field of work, for example mathematics or physics
  - otherwise the successor to the font set that is now more or less standard, Computer Modern plus *AMS*-Fonts, will contain exotic symbols such as  $\approx$  and  $\sim\sim$ , but not basic ones like the male and female symbols
- one or more persons all speaking the English language

During this conference there will be a panel 'The future of *TEX*'. An important subject, something the TUG board, TUG members and *TEX* users in general should think about a lot. As I said earlier: in principle, *TEX* has great potential for authors in *all* scientific disciplines and *all* languages, but only if the program is developed further.

## Future of Publishing

The last topic I would like to talk about is the future of publishing. I don't think I am the right person to make prophecies concerning the future of publishing. Instead, I would like to present some ideas I have found in recent science fiction stories and novels.

One of the most striking ideas I've come across in the past couple of years is the idea of direct brain-computer coupling, as used by the Canadian author

William Gibson, who is called the founder of the sub-genre 'cyberpunk', in his *Neuromancer* novels. With the direct brain-computer coupling, you can access any collection of data and it is as if you navigate with a virtual body through the space of data, which Gibson called 'cyberspace'. It is not such a weird idea at all, although an idea of the far future, and it is related to what people call 'virtual reality', a very popular phrase in some circles nowadays.

An idea that might become reality in the near future can be found in a book by the American science fiction writer, David Brin, in his latest novel 'Earth' [1]:

If only it were a modern document, with a smart index and hyper links stretching all the way to the world data net. It was terribly frustrating having to flip back and forth between the pages and crude flat illustrations that never even moved. Nor were there animated arrows or zoom-ins. It completely lacked a tap for sound ... in a normal text you'd only have to touch an unfamiliar word and the definition would pop up just below. Not here though. The paper simply lay there, inert and uncooperative.

To leave fiction and come back to the here-and-now: according to the Faxon Planning Report 1992 [3], Faxon Press<sup>1</sup> poll of 52 periodical publishers, half of them commercial publishers, the other half non-profit organizations, a small majority of these publishers were quite worried about the future of publishing as we know it. Almost all of them still believe in the primacy of printed books and journals for decades to come. Is the vision David Brin presents something of the very far or of the very near future?

Just a few points to think about:

1. There are still librarians and scientists who see nothing whatsoever in electronic journals and books.
2. But the amount of information printed on paper increases exponentially.
3. And finding the right information becomes increasingly difficult.
4. Furthermore, increase of paper usage is also a serious environmental problem.

Well, you can't halt progress: electronic books are here already and their number will grow. In the transition period there is still another problem. An

---

<sup>1</sup> A large, completely automated subscription agent in the United States, involved in many activities.

electronic book has to be available in paper form as well, since most readers still prefer a paper book.

Suppose you use  $\text{\TeX}$  for the paper version, what do you use for the electronic version? How do you handle the two presentation styles? This is something I hope John Lavagnino will address in his talk on simultaneous electronic and paper publication of Thomas Middleton's complete works.

Is DSSSL<sup>2</sup> the answer to these problems, or FOSI<sup>3</sup>? What will the role of  $\text{\TeX}$  be in non-paper publishing? I really don't know, but we should all think about it.

$\text{\TeX}$  is superior compared to desktop-publishing programs. It can handle mathematical formulas and complex tables, and this is a capability that is often lacking or poorly developed in desktop-publishing programs. Existing programs for the creation of electronic books also lack these capabilities: they can handle only text and graphics. If you want to include mathematical formulas or tables, the most sophisticated you can do is prepare bitmaps of these components — by means of scanning, or perhaps  $\text{\TeX}$ ? — and put these in the electronic document in the form of graphical objects.

## Conclusion

This conference offers a great opportunity for discussions between  $\text{\TeX}$  users and commercial professionals, since the programme contains a lot of talks about many different current applications. There are interesting panel discussions and hopefully there will be plenty of time for discussions during the breaks and in the evenings.

One of the goals of this conference is to try and bridge the gap — apparent or real — between the two poles of my first dichotomy: the author who is a  $\text{\TeX}$  user, and publishers or other commercial professionals who want to accept  $\text{\TeX}$  material. Looking at and thinking about present applications of  $\text{\TeX}$ , as well as an historical perspective, can help to bridge this gap.

This conference is also a good opportunity to discuss the future of  $\text{\TeX}$ , the future of publishing and the future of  $\text{\TeX}$ -in-publishing. And I hope that it will be a success in all respects: that we will be able to find solutions to the problems I mentioned

<sup>2</sup> An ISO standard under development for the specification of document processing, such as formatting and data management [7]. The acronym stands for 'Document Style Semantics and Specification Language'.

<sup>3</sup> See the paper by Andrew Dobrowolski in these proceedings.

and those that will be described in the next four days — that we will be able to make gates in the fences and not just sit on the fences.

I'd like to thank the organization for inviting me to give this introductory talk. It was a pleasure to prepare and give this talk, and I feel honoured having been invited here.

## References

- [1] Brin, David. *Earth*. New York: Bantam Spectra, 1991. [Quoted with permission from the author.]
- [2] Bryan, Martin. *SGML: An Author's Guide to the Standard Generalized Markup Language*. Workingham, UK: Addison Wesley, 1988.
- [3] *The Faxon Planning Report 1992*. Boston: The Faxon Press, Boston, 1991.
- [4] Herwijnen, Eric van. *Practical SGML*. Dordrecht: Kluwer Academic, 1990.
- [5] Humphreys, Christmas. *Zen Buddhism*. London: Unwin, 1984.
- [6] International Standard ISO 8879: Standard Generalized Markup Language (SGML). Geneva, 1986.
- [7] International Standard ISO 10179: Document Style Semantics and Specification Language (DSSSL). Geneva, 1991.
- [8] Mittelbach, Frank. "E- $\text{\TeX}$ : Guidelines for Future  $\text{\TeX}$  Extensions." *TUGboat* 11(2), pages 337–345, 1990.
- [9] Sperberg-McQueen, C.M. "Specifying Document Structure: Differences in  $\text{L}\text{\TeX}$  and TEI Markup." See elsewhere in these proceedings.
- [10] Sperberg-McQueen, C.M., and Lou Burnard, eds. *Guidelines for the Encoding and Interchange of Machine-Readable Texts*. Chicago and Oxford: Text Encoding Initiative, November 1990. [Available upon request from the authors.]

# Comparing T<sub>E</sub>X and Traditional Typesetting for the Composition of a Textbook

Laurie J. Petrycki

Addison-Wesley Publishing Company, 1 Jacob Way, Reading, MA 01867

617-944-3700

[crw@wjh12.harvard.edu](mailto:crw@wjh12.harvard.edu)

## Abstract

Producing a textbook with T<sub>E</sub>X, as opposed to a traditional typesetting system, requires different procedures to achieve a similar final result. The publisher's production staff takes on a much different role and enters the publishing process at an earlier stage when a book is produced with T<sub>E</sub>X. The most significant issue Addison-Wesley faces when a book is typeset with T<sub>E</sub>X is the availability of typesetting houses who can produce the book at the level of typographic and page make-up quality we require. When we use a traditional typesetter we may pay a higher price, but we can count on meeting our publishing standards. The most significant advantage in producing a book with T<sub>E</sub>X is the accuracy of mathematical material, which then does not have to be rekeyboarded, and with which we can easily produce a subsequent edition or spinoffs.

## Background

Addison-Wesley Publishing Company is primarily an educational and technical publisher. The Higher Education Division publishes approximately 100 titles per year in the following disciplines: computer science, engineering, business, economics, physics, and mathematics. The complexity of these 100 titles varies greatly—from one-color, sparsely illustrated books to four-color, heavily illustrated, and designed books. Approximately twenty percent of these books are produced with T<sub>E</sub>X or L<sup>A</sup>T<sub>E</sub>X.

The publishing process begins when an acquisitions editor signs a contract with an author. After the manuscript is written and reviewed by the author's peers, the project is officially turned over to the production department. A production supervisor is assigned to shepherd the manuscript through the production process, with the end result being final film that can be sent to a printer for printing and binding. The production process consists of designing, copyediting, preparing the manuscript for typesetting, rendering the art, typesetting, proofreading, checking galleys and page proofs, and final film.

Like most other production departments within large publishing houses, much of the hands-on portion of the production process—the copyediting, design, art rendering, and typesetting—is done by outside vendors. The in-house staff consists of generalists (production supervisors) who coordinate the project from start to finish, and specialists who arrange for purchasing technical art, typesetting, and cover designs. Addison-Wesley has added a special group to its in-house staff called electronic production. This group (of which I am a member) is responsible for all projects that are produced in a nontraditional manner, which includes T<sub>E</sub>X. Addison-Wesley is committed to staying on the cutting edge of production technology and recognizes the necessity of such a group to consult with authors and the rest of the division.

To understand how T<sub>E</sub>X typesetting affects the traditional publishing process I want to first describe traditional procedures and then compare it with the T<sub>E</sub>X publishing process.

## The Traditional Production Cycle

**The author's role.** The author creates the manuscript, generally using a word processor, and is

responsible for providing a doubled-spaced manuscript to facilitate copyediting. The author is given guidelines for preparing the manuscript—this includes preparing art sketches; following Addison-Wesley editorial styles; placing of figure captions, footnotes, and references; numbering of heads, equations, figures, and tables; and, later in the process, creating an index.

If the manuscript includes mathematical expressions, the author must either leave space on the hard copy and insert the expressions by hand, or use a technical word processing package that can represent the expressions. Technical word processing packages, however, are usually limited in their choice of special characters and their ability to represent complex built-up equations. Thus, an author frequently has to insert some material by hand each time the manuscript is printed out.

If the manuscript has been created with a word processor, we make every effort to use the word processed file rather than rekeyboard the entire manuscript. Traditional typesetting systems can interface with a variety of word processing packages, and some of them can preserve formatting, such as boldface, italics, and tabs. Mathematical expressions in a word processing file, however, cannot be converted to a traditional typesetting system. Math in a word processing program or in *TeX* is coded differently from math in a traditional typesetting system. Invariably, the math expressions, as well as any computer program listings and tabular material, must be rekeyboarded.

After the author has submitted the manuscript to the publisher, his or her role consists of verifying and checking proofs throughout the production process. The author sees: the manuscript after it has been copyedited; the art after it has been rendered; and the galley and/or page proofs after the manuscript has been typeset. If the book has mathematical expressions or computer program listings, the author must pay particular attention to proofreading this material since it is rekeyboarded by the typesetter.

**The publisher's role.** When a book is produced traditionally, the production department is minimally involved until the manuscript is nearly written. At that point, the acquisitions editor holds a meeting with the production supervisor, conveying the market needs for the book, the desired budget and schedule, and any special quality considerations. The production supervisor commissions a design and lines up outside vendors such as a copyeditor and a proofreader.

After the manuscript has been copyedited and typemarked, it is sent to a professional typesetting house for keyboarding and formatting. The typesetter has already received the design specifications and has written a program to interpret the designer's specifications into their coding system. If the book is relatively simple, the typesetter may output the text directly to a final paged format. If the book is complex (for example, if it has a high frequency of illustrations or special design elements), the typesetter will output galleys first.

Galleys are lengths of unpaged, but formatted type. The galleys are proofread and then dummied into pages. By this point the art has been rendered, so the dummier can lay out the galleys on a page grid and indicate where the illustrations should fall. Dummying is an exacting and critical skill that cannot be replicated by automatic pagination programs. In determining where the illustrations fall, the dummier evaluates each double-page spread of the book, looking forward or backward through as much as an entire chapter, to ensure that the illustrations flow evenly and do not fall more than one page past their reference.

The dummier must adhere to certain paging standards. The most important of these standards is that each double-page spread must align across the bottom of the pages. To do this, the dummier has the flexibility of manipulating the space above or below design elements, such as heads, boxed material, illustrations, and equations. However, the dummier must ensure that the space around these elements is consistent across the double-page spread. The dummier must also make sure that there are no widows, orphans, or pages ending with hyphenated words, and that figures and tables are not stacked.

Pages are made up from the dummy, either by hand from corrected galleys or on the typesetter's paging system. The typesetter outputs page proofs which are checked again by the publisher and author. Once the pages are corrected, final film is made, with the art film stripped into place.

## The *TeX* Production Cycle

When *TeX* or *LATeX* is used, most of the above steps of the traditional publishing process are altered. (The issues discussed below refer to manuscripts prepared with both *TeX* and *LATeX* unless *LATeX* is mentioned specifically.) *TeX* disturbs the linearity of the traditional publishing process by forcing the author and publisher to change their roles and by changing the sequence of key events. These changes

are not necessarily detrimental; rather, they make it all the more important for the publisher to define the author's responsibilities and to determine a production plan for each project early on.

**The author's role.** The first change for the author is being put in contact with the production department *during* the contract signing process instead of *after* the manuscript is written. The choice of TeX as the typesetting system for a project is usually driven by the author's desire to use TeX and his or her expertise. As consultants to authors and acquisitions editors, the electronic production group helps determine the level of involvement the author will have in the production of his or her book. An author is rarely encouraged to use TeX unless the author has used TeX previously and is comfortable with the coding process.

Secondly, an author plays a more significant role in ensuring the accuracy of the material and in controlling the schedule. Since the manuscript is not going to be rekeyboarded by a traditional typesetter, it is even more important that the author's initial keyboarding be free of errors. Once the file has been paged, the cost of correcting errors is greater.

Finally, the author gets involved in the type formatting process, which used to be the responsibility solely of the traditional typesetter. This involvement varies according to the production plan. In some cases, the author may be responsible for providing final camera-ready copy to the publisher. More often the author does the initial formatting using TeX macros, then turns the files over to the publisher for final formatting and paging.

**The publisher's role.** With the author taking over more responsibility for the publishing process, it becomes even more critical that the author's and publisher's roles be defined during the contract stage. In general, the publisher retains control over the interior and cover design, the copyediting, the quality of the page makeup, the artwork, and the final filming. However, this is not always the case, as an author may have been given the responsibility for turning over completely camera-ready copy. Assuming these responsibilities are still within the control of the publisher, the TeX production process departs from the traditional process in the following ways.

First, the roles of the publisher's production staff are changed. To produce a TeX typeset book the publisher needs typesetting specialists who are familiar with both TeX and the book publishing process. The traditional production

route is well established and straightforward for the typical production coordinator. However, the influx of word processing files, from a variety of programs, has led to a need for specialists within the publishing house to help with the conversion and smooth translation from electronic file to the compositor or freelancer.

Secondly, TeX reverses the traditional order of the publishing process. Most notably, the production department must start working on the book design before the manuscript has been written, whereas in the traditional model a design is usually not done until a manuscript is nearly final. When the design is done after some of the manuscript is written, the designer has the advantage of looking at the manuscript's elements, making sure that the design is appropriate for all situations. For example, the designer will look at the shortest and longest instances of a chapter title and design the title accordingly.

When TeX is used, the designer does not have this advantage and instead must approximate the final manuscript. Invariably this means that some follow-up design must be done, as new elements are added to the manuscript or if the manuscript structure does not fit the design.

The traditional order is also affected during the copyediting and art rendering stages. On a traditional manuscript the copyeditor not only edits but marks up the manuscript for the typesetter by indicating the various type elements. With TeX this step of typemarking is incorporated into the initial formatting the author does. As the author chooses a particular macro for a text element, he or she is essentially doing the copyeditor's job of typemarking. However, the copyeditor must check to make sure the author has used the correct macros.

Art rendering is traditionally done while a manuscript is being copyedited and set into galleys. Since there isn't a comparable galley stage in the TeX production process, the art rendering stage is on a tighter schedule.

Finally, TeX introduces new types of outside resources to the publisher. After the TeX manuscript has been copyedited, it is paged, with space allowed for the art to be added or the art merged electronically. This task is contracted out to a typesetter who specializes in TeX or L<sup>A</sup>T<sub>E</sub>X composition. There are not many of these typesetters to choose from, especially ones who are experienced in our exacting textbook standards and who can manipulate TeX. Theoretically the final product should look the same, no matter which production process

is used, so it is important that the appropriate typesetter be chosen.

**Publisher-assisted formatting.** To expedite the  $\text{\TeX}$  production process we find it advantageous to supply our authors with comprehensive macro packages as early in the manuscript writing stage as possible. These macros follow one of several book designs and have been thoroughly tested and documented. The author is responsible for the initial formatting of the manuscript with these macros, but at some point we take over the follow-up stages. We call this process publisher-assisted formatting, and our goal in working this way is to retain as much of the author's formatting as possible, while allowing the author to concentrate on writing the manuscript.

Many authors, especially those who are using  $\text{\TeX}$  for the first time, tend to rely on plain  $\text{\TeX}$  commands instead of macros when they set up the design parameters. For example, an author might type out the command string

```
\bf, \medskip\item\vskip2in\hskip1em
```

each time an item in a numbered list is called for, instead of incorporating the same string into a macro. Whereas plain  $\text{\TeX}$  commands get the job done, design changes are difficult to implement throughout the file. Using macros allows us to make these design changes globally and, in general, allows for smoother formatting when the files are sent to a typesetter. Therefore, we ask all authors to avoid using plain  $\text{\TeX}$ , except within mathematical expressions. Although there is always the danger that any customization of the math spacing, using plain  $\text{\TeX}$  commands, will interfere if the final book is reformatted in a different typeface or size.

When we give an author a macro package, we include a set of instructions for using the macros, as well as guidelines for paging the book. We tell the author how to set up files in order to avoid redundant effort—for example, an author should not put in manual page breaks, because the composition of the pages will change several times before the book is complete. We also show authors how to add any custom macros to the macro file, and we stress that authors code their files consistently.

Providing authors with professionally designed macro packages and then contracting with a typesetter to finish the formatting allows us to control the quality of the finished product. However, there are instances when an author has contractually agreed to provide camera-ready copy. In these cases, we provide guidelines for paging the book to

our publishing standards and we follow the process rigorously, acting as quality checks for the author along the way. Most authors who are proficient with  $\text{\TeX}$  have used it to write class notes, papers, or perhaps journal articles. They do not realize how much more difficult and time-consuming it is to make  $\text{\TeX}$  conform to textbook standards.  $\text{\LaTeX}$  is even more difficult to use because its sizing and figure placement features must be overridden.

The macro package we give an author may not necessarily reflect the final design, which can change for a number of reasons. We may have predetermined that the typeface will be changed from Computer Modern and the author may not have the new typeface available. Or the design may not be determined, but we don't want to delay the author's writing. In any event, the beauty of  $\text{\TeX}$  is that once the file is properly coded with macros, another set of macros can be substituted.

Supplying macros to authors is an ongoing and challenging task. Our authors work on a variety of computer and printer platforms and the availability of typefaces other than Computer Modern further complicates matters. Macros that work well on a test chapter may not work in an unforeseen combination of text elements that an author sets up. An author may want to add a new text element or his manuscript might not fit the design. All of these challenges mean that we must keep a library of macro packages available and must continually update and debug the packages as new fonts or platforms are introduced. In spite of these challenges, we feel that  $\text{\TeX}$  gives us certain advantages.

## Advantages of Using $\text{\TeX}$

**Technical material.**  $\text{\TeX}$  is an easy and efficient tool for representing mathematical equations. The author writes and typesets concurrently, as opposed to having to handwrite equations that cannot be represented in many word processing programs. Equations, computer program listings, and tabular material do not have to be rekeyboarded by the typesetter, whereas with other programs they do have to be rekeyboarded. This type of material is the most difficult to proofread, and authors want to be assured that once they have verified the accuracy of such material, it will not change. Of course, there is the proviso that any time electronic material is converted between different systems or editing changes are added to a file, there is the possibility of errors being introduced. Thus the

importance of thorough proofreading by both the author and publisher is not diminished.

When providing  $\text{\TeX}$  files, however, it is even more critical that the author be accurate in the first place. Correcting mistakes that the author made in the initial keyboarding may be charged to the author or publisher if they were not the final typesetter's responsibility.

**Split and revised editions.** When we decide to produce a book using  $\text{\TeX}$ , we're also looking down the road at future editions, spinoffs, or splits (certain chapters of the book rearranged and/or removed for different versions of the book). Using  $\text{\TeX}$  gives us a strategic advantage for this kind of expanded publishing platform.

Producing split editions from a traditionally typeset book often involves a painstaking update of the references, the table of contents and the index. There is a certain amount of page make-up that has to be redone. With  $\text{\TeX}$ , however, counters and macros can be used to automate these changes.  $\text{\TeX}$  macros can regenerate the table of contents and index files, and update the cross-referencing of figures, equations, and tables. The file is just reprocessed, then output again. No additional page makeup is required.

Producing future editions from a traditionally typeset book is often problematic. In general, a traditional typesetter can download files from their system back to an author's word processing system. However, if there is any math or tabular material, this process will not work. The typesetter's coding structure around this material cannot usually be converted to the author's system. Also, the traditional typesetter must ensure that the final version of the files have been updated, including last minute or reprint corrections. Often this does not happen—a traditional typesetter will set these corrections as one-line patches, rather than update and run out the entire chapter file.

This is where  $\text{\TeX}$  shines. For future editions, it is easy and economical to return  $\text{\TeX}$  files to an author, with the codes intact. The files are accessible to the author for the revision and can be uploaded back to the  $\text{\TeX}$  typesetter. Even if we do not know the final design of the book, we can provide an author with a generic macro package to produce a coded file. Later the design can be completely modified, and the  $\text{\TeX}$  typesetter merely rewrites the definitions of the macros. If the original macros were planned and used well, the final design can be implemented with a minimum of additional coding.

**Cost and schedule advantages.** In general, the composition portion of books produced with  $\text{\TeX}$  are less expensive and take less time to produce than traditionally typeset ones, but there are several caveats here. If a  $\text{\TeX}$  file is inconsistently coded to begin with, it can cost us more to clean it up than if the book were traditionally typeset. Also, the author's initial writing time may increase substantially, even adding a year or more to the schedule, if the author gets bogged down in the formatting. These factors have to be weighed in the initial evaluation of each project.

Even so, when  $\text{\TeX}$  works out, it allows us to publish in limited markets we normally would not be able to publish in. These markets, such as graduate level courses, can only support small print runs of a book, thereby decreasing the book's profitability. Using  $\text{\TeX}$ , we can keep our production costs down.

**Satisfied authors.** An author who uses  $\text{\TeX}$  feels more assured knowing his or her keystrokes will be used in typesetting the final book.  $\text{\TeX}$  is almost necessary when the author needs to represent complex and frequent mathematical expressions. Authors who want a lot of control over the layout of the pages want to work with  $\text{\TeX}$  to provide final  $\text{dvi}$  files or camera-ready copy. The author must trade off that satisfaction by spending a considerable amount of time formatting his or her manuscript.

## Disadvantages of Using $\text{\TeX}$

**Steep learning curve.**  $\text{\TeX}$  is not easy to learn, especially if an author has not had previous experience with the program or is not familiar with a code-intensive computer language. Most authors who have not used  $\text{\TeX}$  before are familiar with WYSIWIG (What You See Is What You Get) types of word processing programs, which are generally easier to use. Thus, trying to learn  $\text{\TeX}$  while on a fixed schedule to produce a manuscript can be overwhelming. In general,  $\text{L}\text{\TeX}$  is easier to learn for the new user.

In spite of this challenge, some authors will learn  $\text{\TeX}$  for the first time because they want to retain more control over their technical material or because an acquisitions editor may require a camera-ready manuscript. The danger here is that the author can easily spend more time learning how to format with  $\text{\TeX}$  than actually writing the manuscript.

**TEX paging problems.** There are a number of instances where **TEX**'s automatic paging can present more difficulties than it solves.

1. When **TEX** pages, it does not recognize double-page spreads—instead it processes one page at a time. Aligning the bottoms of pages across spreads is critical to textbook design. In traditional typesetting the operator or dummier can review several pages at a time, going backwards and forwards through a group of pages to allow for the best possible layout of each page. **TEX** only considers the current page, so we must manipulate **TEX** to give us the required results.
2. **TEX** does not always pull out the most current first-level head as the running head. The number one head is usually pulled out by the second occurrence.
3. Figures can occasionally appear out of order, especially if the file has a lot of figures. **LAT<sub>E</sub>X** has a tendency to lump several of the figures at the end of a chapter file or place them too far away from their page reference. **LAT<sub>E</sub>X** also tends to add a lot of additional space below figures, although this is adjustable manually.
4. Minor edits or revisions which should be confined to the current page or paragraph sometimes affect subsequent pages or the spacing around elements. Not only does the typesetter have to reprocess the entire chapter, but the production supervisor must thoroughly recheck subsequent pages, rather than just the one line where the change occurred. This creates a great deal of additional and unnecessary work for each proof stage of a book. Often we will have to manipulate the **TEX** coding so that the pages match the previous output. In a traditionally typeset book, such changes are always limited to the page on which the change occurs.

This problem was compounded on a recently reprinted two-color book. Many of the reprint changes were simple typographical errors that should have affected only one line in a paragraph. In fact, the page breaks did not change on those pages and the black text pages looked fine. However, the spacing between all elements on the page changed a small amount, which was not noticeable until the film for the black text was no longer in register with the film for the second color. We had to reshoot and restrip all the second color film to make it align with the revised black text page.

**TEX math spacing.** For the most part, the internal spacing around math characters in **TEX** poses no problems. We have, however, uncovered two instances where we feel the spacing is unacceptable. These are: the spacing around extensible parentheses and the lack of kerning in sub- and superscripts. Examples of these instances follow.

Notice in the following first equation where the proper size parentheses are used, the space after the open parenthesis is too large. In the second equation where the normal size parentheses are used, although incorrectly sized for the equation, the spacing is correct.

$$\$ \$ u = \left( 1 - \frac{u^2}{u+2} \right) \$ \$$$

$$u = \left( 1 - \frac{u^2}{u+2} \right)$$

$$\$ \$ u = (1 - \frac{u^2}{u+2}) \$ \$$$

$$u = (1 - \frac{u^2}{u+2})$$

Letter pairs are not automatically kerned when used for sub- and superscripts. Notice the spaces between the letters in the examples below.

$$\$ \$ v^{CE} \$ \$$$

$$v^{CE}$$

$$\$ \$ v^{VJ} \$ \$$$

$$v^{VJ}$$

$$\$ \$ v^{To} \$ \$$$

$$v^{To}$$

As mentioned earlier, any spacing adjustments to equations must be done in the correct type and point sizes. Translation problems have occurred when special math coding is done by an author and then the macros are converted to a different typesize and style.

**TEX is not always device-independent.** When using fonts other than Computer Modern, **TEX** is not strictly device independent. Typefaces are designed by different companies to run on different output devices. The character widths in Compugraphic's version of Times Roman differ from each of Autologic's and PostScript's versions. Therefore, a source file processed through one output device will have different line breaks when run through another output device.

This makes it difficult when authors format files using their own computer system and laser printer,

then give us the files to output through a high-resolution typesetter. One solution is to provide authors with *tfm* files that match the character widths on the final output device. The author can still use Computer Modern on his/her laser printer, but each character will be the width of the corresponding font on the high-resolution output device. The disadvantage here is that proofreading can be a challenge. The space between letters on the laser proof will not be accurate—some characters will appear kerned too tightly and others too loosely, when they may in fact be correct—so the proofreader must bear this in mind.

The implementation of PostScript on different devices affects how T<sub>E</sub>X's \special commands are interpreted. Special PostScript effects, such as rotated type, rules, and shadings, can change position depending on the T<sub>E</sub>X printer driver implementation. Other problems can occur because T<sub>E</sub>X and PostScript use a different measurement for points to the inch. PostScript rounds to 72 points to the inch, while T<sub>E</sub>X correctly uses 72.27 points to the inch.

We were most disappointed when we ran out an author's *dvi* files, which were entirely in Computer Modern, and found differences between his laser proofs and the high-resolution output. The more recent PostScript Computer Modern fonts have been being calculated differently. We were told that the Computer Modern font itself is not static but is being revised constantly. This can create major problems for publishers supporting authors on different systems.

## **Issues to Consider When Choosing T<sub>E</sub>X**

In spite of the disadvantages just listed, more often than not we choose to use T<sub>E</sub>X because the pros still outweigh the cons. When we determine the production plan for each T<sub>E</sub>X project, however, there are certain issues we need to determine.

**What is the author's level of production involvement?** We must first define the author's role in the production process, because from that definition comes clarification of the publisher's and typesetter's responsibilities. Some authors will contract to provide camera-ready copy, including rendering the art, while others will do the initial formatting only. No matter which route the author takes, the publisher is ultimately responsible for the overall quality of the book, so it is up to

the publisher to ensure that the production plan includes the necessary checks and balances.

Defining roles is also important when it comes to the cost of making the book. For example, if the author is responsible for inserting the copy edits into the source files, then there are repercussions later on when changes are made during the page formatting stage. Who is responsible for the changes and, more importantly, who pays the typesetter to make the changes?

**Who controls the schedule.** Ensuring a book's schedule is one of our major responsibilities as publishers and when T<sub>E</sub>X is involved there is a great danger of schedule slippage. When the author takes a more active role in the production process, the publisher loses control over that part of the schedule. This is particularly detrimental when authors become overwhelmed by the extent of their responsibilities. Many of our authors have full-time teaching positions and do not initially realize how time-consuming book production is, whereas publishers are used to working with book production professionals who can commit to a 40+ hour week.

For this reason, we have occasionally found it necessary to change the production plan midstream. We have either taken over some of the tasks the author was initially responsible for or have added additional proofreading or checking stages. Our biggest concern here is that an author will become so involved in T<sub>E</sub>X formatting that he or she slows down the writing of the manuscript.

**Quality considerations.** We must determine if T<sub>E</sub>X can give us the quality level we need to publish into a particular marketplace. We have exacting quality standards for books that are produced traditionally and, as stated earlier, when left on its own, T<sub>E</sub>X's formatting does not always meet those standards.

For example, we usually require an equal amount of space above and below displayed equations across a double-page spread. When building a page, T<sub>E</sub>X does not automatically do this. It takes either a considerable amount of manual manipulation or a complex rewrite of T<sub>E</sub>X's macros to achieve the proper spacing.

Another concern is that T<sub>E</sub>X's glue often stretches or shrinks erratically unless the macro package is expertly written to account for this variability. To balance a double-page spread when a book is typeset traditionally, we can specify exactly how much extra space to add at particular points and the typesetting program will follow our specifications. This is also possible with T<sub>E</sub>X, but it is

an added challenge that only the most experienced  $\text{\TeX}$  typesetters can handle.

**Beyond Computer Modern.** For most textbook publishers, Computer Modern Roman is not an aesthetically pleasing basal (main text) typeface. We feel that the x-height of the letters is too small and the typeface looks old-fashioned. Because we publish into a variety of college markets, we need to have the option of different typefaces. For lower-level textbooks, we use more open, friendly typefaces like Century Schoolbook; for upper-level textbooks we use more sophisticated, professional typefaces such as Times Roman. Some  $\text{\TeX}$  compositors can offer us Times Roman and other standard typefaces as replacements for Computer Modern in the basal text. However, only a select few compositors can also typeset math equations in Times Roman or other typefaces. Converting the  $\text{\TeX}$  math character set takes a lot of programming time and expertise—and few compositors are willing to make this investment.

Some  $\text{\TeX}$  sources will offer the compromise solution of mixing Times Roman or another standard face for the basal text and displayed material, such as heads, with Computer Modern math. This solution is not perfect, however. The weight of Computer Modern and Times Roman characters is different—Computer Modern is smaller and lighter than Times Roman—so the resulting mix looks odd. To make the two typefaces appear uniform, Computer Modern must be increased by one half point.

One last note on typeface substitution— $\text{\TeX}$  kerning and ligatures do not always work on typefaces other than Computer Modern unless the programmers who have written the output drivers have done the extra work to provide the conversion. On some implementations of PostScript fonts in  $\text{\TeX}$  --- will give you three dashes instead of the correct emdash.

**Working with specialized  $\text{\TeX}$  typesetters.** Finding the appropriate  $\text{\TeX}$  typesetter is one of our major challenges. We need sources skilled not only in programming  $\text{\TeX}$ , but also in typesetting textbooks rather than journals or papers. The quality level of paging is more exacting for technical textbooks and traditional typesetters have a solid typographical and book-making background. These typesetters are used to dealing with publishers' demanding schedules, and usually have larger staffs to call on.

We know what kind of service we will get when we work with traditional typesetters. Unfortunately,

with some  $\text{\TeX}$  typesetters this is not always the case. When we receive galley or page proofs from a traditional typesetter, we can safely assume that they have been proofread, whereas we often have to request this service from a  $\text{\TeX}$  typesetter. The output from traditional typesetting systems is almost always typographically correct. With  $\text{\TeX}$  typesetters, we sometimes have to specify correct alignment and kerning. Finally, we can rely on the typesetter for meeting our agreed-upon schedule and can even request overtime to meet a tight schedule. Since some  $\text{\TeX}$  typesetters have small staffs, we have often run into schedule overloads at their end.

## Conclusion

In my experience, traditionally produced books are more predictable and easier to work on than those produced with  $\text{\TeX}$ . However,  $\text{\TeX}$  does have its place in the technical publishing house. For some authors, using  $\text{\TeX}$  is the most viable option when they want to preserve the accuracy of their mathematic equations. We will continue to support these authors by providing macro packages and working with  $\text{\TeX}$  typesetters to provide the same kind of services we expect from more experienced traditional typesetters. Producing a book with  $\text{\TeX}$  is a process that can proceed as smoothly as traditional typesetting as long as we have done the proper upfront planning and have evaluated the tradeoffs.

# Contra-LATEX, or What Really Works in the Publishing World

Frederick H. Bartlett

The Bartlett Press, Inc., Harrison Towers, 6F, 575 Easton Avenue, Somerset, New Jersey 08873 USA  
908-745-9412  
CompuServe: 72450,2574

## Abstract

An only slightly cynical view of the real interactions among authors, publishers, and TEXnicians.

## Introduction

My purpose in this article is to describe, as honestly as I can, how TEX is and should be used in what we sometimes like to call the Real World (although those of you who have actually dealt with publishers may question the validity of that appellation).

Since few of my readers will know me, I feel that I should give a brief account of myself. I have been a technical writer for a small computer company; a production editor for a series of proceedings; an acquisitions editor for an international scientific publisher; and, for the past six years, the head of a TEXnical typesetting and production house. Thus, I have some experience of every part of the process of publishing, from the time a writer gets an idea or an assignment to the time the finished product is sent to the bookstores.

My company is one of the very few commercial typographers to use TEX for all its typesetting tasks, from initial keyboarding to final layout. As far as I can judge from advertisements in *TUGboat*, there are fewer than a half dozen similar firms, although there are many individuals and organizations which use TEX in some way, whether writing macros or providing output services.

Most TEX users, however, are salaried employees of commercial or educational organizations; as their incomes are not directly determined by the number of pages they are able to produce per day, and as their employers, not being publishers, are not concerned with the niceties of typographic style, our concerns—speed, efficiency, quality—are not necessarily theirs.

This undoubtedly explains the otherwise mystifying popularity of LATEX.

## The Promise of TEX

For ten years or more, TEX has promised authors full control of the typographical appearance of their books and publishers a way to turn out high-quality

books at a much lower cost. Unfortunately, the two promises too often remain unfulfilled.

First, authors, as a class, are completely ignorant of what Thomas Browne calls “the Trade and Mystery of Typographers.” Second, publishers are not interested in producing high-quality books; they are interested only in producing books that look good enough to sell. Many of you may have seen the article by Jacob Weisberg in the June 17 *New Republic* on the lamentable state of trade publishing. More personally, just before I left the editorial department of an international science publisher, I was reprimanded by the chairman because, as he put it, my standards were too high.

This is not to say that authors are idiots and publishers Scrooges; merely that an author’s first concern is the information he’s conveying and a publisher’s first concern is the money he’s making (or, more often, losing). It is clearly senseless to require authors to be typographers or publishers philanthropists—it’s nice when it happens, though.

The result, however, is that most books produced with TEX are easily identifiable by their shoddy appearance.

## Commercial TEX

In order for TEX to take what I believe to be its rightful place as the typographic language of choice for books and journals, more typesetting firms must adopt it and more production departments accept it. To illustrate how far we are from such a state, let me tell a more-or-less fictionalized little story.

Someone from TEXnical TEXtbooks Limited (we’ll call him Fred) calls the production director of Acme Worldwide Publishing Co., Inc. Assuming that he perseveres through phalanxes of secretaries and assistants, he might say, “Hello. I represent TEXnical TEXtbooks Limited, a TEX-based typesetting firm. We can satisfy all your typesetting

needs, especially if you get books in electronic form prepared using **T<sub>E</sub>X**."

Now, the production director (we'll call her Ms. Constant Tradition) will say one of four things: (1) "We are perfectly happy with all our current vendors" (this is the usual response), (2) "We prefer not to use desktop publishing firms," (3) "We don't publish technical books," or (4) "We don't use cottage industry-type vendors."

Assuming that he got one of the latter three responses, Fred will try (usually in vain) to convince Ms. Tradition that (a) **T<sub>E</sub>X** is not "desktop publishing," (b) **T<sub>E</sub>X** can typeset anything, and (c) the "technological cottage" approach will save her money.

Now, why is Fred having such trouble? We will charitably discount the possibility that he is a lousy salesman. The primary reason is that most publishers' experiences with electronic publishing have been unhappy ones. If you have a trained eye, you can go into any bookstore and determine which books were typeset using DTP software—they're the ones whose appearance ranges from loathsome to just barely good enough to get by. Even most **T<sub>E</sub>X**-set books do not measure up to any but the most minimal of standards. Therefore, production directors don't want to use electronic production techniques unless they absolutely have to, as when they're constrained by the budget or by the contract the editorial department signed with the author (which they will resent like blazes).

If Fred is lucky, he'll be able to send Ms. Tradition a sample book typeset with **T<sub>E</sub>X**. Perhaps he can even send her two books, say, a novel and a mathematical monograph, just to show **T<sub>E</sub>X**'s range. But even this may not convince her to hire him.

For production departments have an unreasonable prejudice against small shops (and most current **T<sub>E</sub>X**—and, it must be confessed, DTP—shops are small). Publishers routinely use one-man shops (called freelancers) to do design, copy editing, and proofreading, but somehow typesetting must be done by large firms with hundreds of employees, huge overheads, and high prices. This problem is, of course, beyond the scope of this paper, but I hope to warn budding entrepreneurs of the problems they're headed for.

Even assuming that Ms. Tradition has been impressed by Fred's presentation thus far, she's unlikely to send him a manuscript to set; instead, he'll get a set of author's disks. Fred will then have the unenviable task of explaining why typesetting

from disks saves 10 to 50%, instead of 50 to 90%, of the cost of typesetting from paper.

There are many reasons for this, but they all boil down to one: the author.

## TEXnical Difficulties

It is an ancient joke among editors that their job would be a real pleasure if it weren't for authors. It is this attitude that explains why authors find themselves completely shut out of the decision-making process once the contract is signed and the book is delivered into the publisher's hands.

It may be that widespread use of electronic document preparation technologies like **T<sub>E</sub>X** may change this attitude, but it is unlikely, since authors have more important things to do than learn the language, techniques, and requirements of fine typography.

For reasons completely opaque to the present writer, **L<sup>A</sup>T<sub>E</sub>X** is the **T<sub>E</sub>X** tool of choice for half or more of all writers who use **T<sub>E</sub>X**. Why in the world, to borrow Dr. Lamport's metaphor, would someone voluntarily exchange a high-performance racing car for a beat-up old family sedan?

Thus, in order to undo what might be called **L<sup>A</sup>T<sub>E</sub>X**'s sedanification of **T<sub>E</sub>X** and create a professional product, the macro writer must spend much more time (and therefore money) than a publisher is likely to consider appropriate. For **L<sup>A</sup>T<sub>E</sub>X** imposes several severe penalties upon its users.

First, a **L<sup>A</sup>T<sub>E</sub>X** file will be 10% or more larger than an identical **plain.tex** file. Keyboard macros are, at best, only a partial solution, and, in any case, cannot be standardized among keyboarders who each use their own favorite word processor or text editor for data entry.

Second, it takes longer to run **L<sup>A</sup>T<sub>E</sub>X**, both on each part of a book and, most importantly, on the entire book, especially since **L<sup>A</sup>T<sub>E</sub>X** assumes that one will process an entire book at a time. Even when one uses an extremely fast computer (we use a 25-MHz 486 machine which can process a 27 × 42 pica page of **plain.tex** in under a quarter of a second), this is a tremendous handicap at the final stages of a job when one is trying to find and set the best page breaks in accordance with the publisher's style. The only solution I have found is to run **L<sup>A</sup>T<sub>E</sub>X** on the entire book twice, save the **.aux** file, divide the job into several parts, and **\input** the **.aux** file at the beginning of every part of the job. Once all the page breaks are set, we then run **L<sup>A</sup>T<sub>E</sub>X** twice more on the entire book, hoping that any

changes in cross-referencing will not affect the page breaks.

Third, inputting corrections, both from the copy editor and from the author and editor, becomes much more difficult. When we set a manuscript using `plain.tex`, we enter the equation numbers as numbers, so that, when we have to add a minus sign to equation 9.34.2, we can search for that equation number, find it quickly, and make the change. If we have a LATEX file to contend with, we must either know the author's `\label` (an unlikely possibility) or search for some unique combination of words or mathematical symbols, such as `\root n \of{-\lambda}`, a penalty of 15 keystrokes and a bit of thought. Thought is very time-consuming, and therefore, as all production editors know, typesetters have always sought to do as little of it as possible.

Fourth, implementing the publisher's style is much more difficult to do on top of LATEX than `plain.tex`. Recently, for example, I received a call from one of a client's authors asking me how to change the length of a page in LATEX. He had been trying various machinations with no success for about a week. Once I received his files, I solved his problem in something under a minute. However, I have never received such a basic query from any author using `plain.tex`.

I have wasted so much (unbillable!) time trying to make LATEX behave that I finally decided to convert whatever LATEX projects I get to `plain.tex`, a process that takes less than an hour, and then write a `plain.tex` macro package. This has the additional advantage of enabling us to use our own output routine instead of LATEX's, so that we can be sure of placing the vast majority of the floating insertions properly the first time through.

I usually keep the few LATEX macros I have found to be both an improvement over `plain.tex` and impossible to convert: the `array` and `tabular` environments.

This is not to say that one cannot produce good-looking books with LATEX, only that it will take longer and cost more. Truth to tell, however, the only LATEX book I've seen that looked decent is *Introduction to Algorithms*, which was published by the MIT Press and McGraw-Hill. Amy Hendrickson provided the LATEX macros. It should be said, however, that the MIT Press's style makes life much easier for the TEXnician and layout person, as it uses ragged bottoms.

If LATEX is such a mess, you may ask, why would anyone, even an author, use it? The usual reasons given are ease of use and standardization. But both

are illusory. LATEX is no easier, and in some ways more difficult, to use than a special-purpose set of even moderately well-designed `plain.tex` macros. And standardization is not helpful unless every format in which a given file is to appear is the same width. (If the widths are different, or if there's a change of point size, all wide alignments and displays will have to be altered manually anyway; this is a far more time-consuming task than letting a few macros to some other definitions.)

## What Is to Be Done?

The easiest way to keep costs down and ensure that production will move as quickly as possible is simply to use `plain.tex` instead of LATEX.

However, authors who use `plain.tex` are—returning to the famous Lamport analogy for a moment—often discovered to be truck drivers merely masquerading as sports car enthusiasts. One of my favorite masqueraders was the author who used his own definition of `\section` for every level of head from chapter openings to subsubsubheads. Others will begin paragraphs in display math mode or end display math mode with two carriage returns and a `\noindent`. However, even a novice TEX user can produce perfectly acceptable files if he keeps a few simple rules in mind.

Of course, it could be said that I am arguing against my own best interests. So long as authors use LATEX and misuse `plain.tex`, there will be a need for TEX wizards to create silk purses out of sow's ears, and I can always charge more for working from LATEX than from `plain.tex`. But I have a Puritan objection to redoing what should have been done right the first time, even if I am being paid for it.

The first rule is to avoid using TEX primitives, especially those which control spacing (`\kern`, `\vskip`, `\hskip`), but always call them from macros (like `plain.tex`'s `\bigskip` etc.). `\vfill`, `\eject`, `\break`, etc., should also be avoided, as should explicit font calls in headings.

It is really not too much to say that the only place an author should use plain or primitive control sequences is in math mode, for the real power of TEX consists in this: that all things are susceptible of change.

The second rule is to use a macro for every typographical or logical entity in your work. Examples are `\section`, `\subsect`, `\list`, `\example`, and `\theorem`. You need not define them, except as, say,

```
\newcount\sectcnt
\def\section#1{\par\global\advance
  \sectcnt@ne \the\sectcnt. #1\par}
or
\def\section#1#2{\par#1. #2\par}
or even
\let\section\relax
```

What about cross-referencing, you may ask. L<sup>A</sup>T<sub>E</sub>X's cross-referencing system is, perhaps, the feature of L<sup>A</sup>T<sub>E</sub>X that authors like best, even if it does pose problems for those who have to deal with the file after the author is finished. But cross-referencing is not difficult; the only advanced T<sub>E</sub>Xnique one needs to know is the `\csname ... \endcsname` primitive.

Those who are not yet convinced that L<sup>A</sup>T<sub>E</sub>X is so awful may wish to emulate a set of macros I once wrote to allow for automatic numbering and cross-referencing. I added one small, but important, function: the characters used as the label appeared in the margin on the proof copies. This could be added to L<sup>A</sup>T<sub>E</sub>X easily enough, but no one seems to have thought of it, as it is universally assumed that only the author is going to have anything to do with the creation of the document.

In the best of all possible worlds, the publisher would arrange for a T<sub>E</sub>X consultant to write macros for the author as he writes his book. Given the way the publishing business works—especially given the traditional hostility between publishers' editorial and production departments—this is unlikely in the extreme, although it would provide publishers with the savings they have always expected from electronic production.

Speaking of money, authors should know that traditional typesetting costs anywhere from \$8 to \$50 per page, depending on the size of the page, the complexity of the material, and the complexity of the design. If an author does all or most of the work himself, he should ensure that the publisher either pays him a fair price or lowers the asking price of the finished product.

Not that he is likely to have much luck. The rule is, "whatever the market will bear," and so long as most purchasers of professional books pay for them with someone else's money, there will not be much pressure on publishers to lower prices. But there's always some—one of my former employers has become notorious recently for both the enormous amounts he charges for his books and journals and his penchant for suing anyone who criticizes his pricing policies.

## Postconference Postscript

**Introduction.** In my preprint, I discussed several books from the standpoint of a critical typographer; as such a discussion has no merit if the readers have no access to the books, I shall here make some general observations about current typographic practices and a few responses to concerns raised by other speakers at the conference.

**Typography Today.** Of the fourteen books I took to the conference for discussion, four were traditionally set, two were set with DTP programs, six were set with T<sub>E</sub>X at The Bartlett Press, and two were set with T<sub>E</sub>X by others.

When one looks at traditionally composed books, one notices that the line breaks are often not as good as T<sub>E</sub>X would produce and that several refinements which used to be taken for granted are now lost. There is one exceptional publisher which still produces extremely high-quality books: The Folio Society. The Society is a subscription publisher devoted to the art of fine bookmaking; anyone confused by prattle about "quality" is urged to examine some of their books.

The refinements I alluded to above include such things as avoidance of widows and orphans, avoidance of recto-to-verso hyphenation, alignment of pages (partly the printer's problem), and alignment of accents over letters.

Books produced by desktop publishing programs typically have lousy layouts (extremely variable space around figures and tables, ragged bottoms, insufficient number of lines below a head), ugly fonts, and an unnecessary, distracting, and ugly proliferation of design elements.

The Bartlett Press's books are, in general, pretty good. The major difficulty we have had is in using non-Computer Modern fonts in mathematics; often the kerning is not ideal. It is, however, quite good enough for most purposes and compares well with the kerning of other math setting systems. Books we have set with little or no math are, for all practical purposes, perfect.

I should confess that, overcome by a spirit of honesty, I brought the first book we ever did, which was produced while we were first learning T<sub>E</sub>X—it had many of the problems I attribute above to traditionally composition methods. Of course, we did learn better. It also proves that someone knowledgeable in typography can get decent results with T<sub>E</sub>X, even though someone trained in T<sub>E</sub>X may produce something typographically awful.

The Bartlett Press often has the advantage of keying its books from MS; books that other companies have set are produced from the author's disks and, usually, on low budgets. It is in these cases that T<sub>E</sub>X becomes a second-rate (or worse) typesetting system. This is especially obvious if the author is his own designer and if he uses only Computer Modern fonts. However, even high-budget books suffer if the T<sub>E</sub>Xnician is insufficiently thorough or insufficiently acquainted with the conventions of typography. For instance, consider the way vertical space is handled when two elements that each contribute space abut one another. Publishers have rigorous standards for such cases, but no standard implementation of T<sub>E</sub>X will perform properly. Of course, T<sub>E</sub>X can handle this problem, but only if the T<sub>E</sub>Xnician is enough of a typographer to do it.

Reading over these comments, I see that they seem a bit churlish and self-aggrandizing. I should say, therefore, that many of our competitors do fine work. Yet it is important that publishers know that there is at least as much variation among T<sub>E</sub>X typesetting firms as there is among traditional firms and, more importantly, that the use of T<sub>E</sub>X does not, in and of itself, guarantee that a project will be either good or shoddy.

**Some Solutions to Some Problems.** Various speakers complained about T<sub>E</sub>X's steep learning curve. But this is a problem only if one wants everyone who uses T<sub>E</sub>X to be a wizard. We train our keyboarders to use T<sub>E</sub>X in a day; after a week they're thoroughly used to it. But how do you handle something really difficult, you may ask. We tell the keyboarders to make up a macro, which they will not even try to define, with as many arguments as they think necessary. When the file arrives in house, we supply the necessary definition. Thus, one only needs one wizard for twenty or thirty users.

Another complaint often voiced had to do with costs and scheduling. A sore point. We cannot guarantee either until we have seen everything pertaining to a job: the complete manuscript, the complete set of files, and the finished design. An estimate based on the first few chapters cannot possibly include the cost of repairing the horrific mess the author made of the eighth chapter. Even so, I am baffled by the assertion that it is often cheaper to have a manuscript reset in the Far East than to have a domestic firm work with the author's files. Our experience tells us that it is a rare author indeed who can make that great a mess of a T<sub>E</sub>X file.

The problem of fonts is still a serious one, but now that virtual fonts are a standard feature of device drivers, the problem will begin to disappear. Meanwhile, users should not be afraid of meddling with T<sub>E</sub>X .pl files to tweak the kerning to their satisfaction. Be very sure, however, to send the resulting .tfm to your output service; otherwise, you will not get very good results.

The problem most often mentioned was that of page makeup. It is undeniably difficult to get T<sub>E</sub>X to set page breaks that uniformly adhere to the publisher's standards. However, creative macro writing can solve all the problems. The simplest case—that of one-column text—is relatively simple, even though no standard set of macros (*plain.tex*, L<sup>A</sup>T<sub>E</sub>X, *AMS-TEX*) can handle it. The general case of multicolumn text is hard; one must do a lot of work to overcome some deficiencies in the design of T<sub>E</sub>X itself.

Given our experience with setting multicolumn material, I suspect that T<sub>E</sub>X will never be widely adopted for newspaper and magazine work unless it is substantially rewritten. This journal (*TUGboat*) is proof enough of that—the design and typesetting are serviceable, but hardly triumphs of the art.

# T<sub>E</sub>X in a Book Production Department

Howard Ratner

Springer-Verlag New York Inc., 175 Fifth Avenue, New York, NY 10010, USA

212-460-1655

Internet: [ratner@spint.compuserve.com](mailto:ratner@spint.compuserve.com)

Kenneth Dreyhaupt

Springer-Verlag New York Inc., 175 Fifth Avenue, New York, NY 10010, USA

212-460-1655

Internet: [dreyhaupt@spint.compuserve.com](mailto:dreyhaupt@spint.compuserve.com)

## Abstract

This paper presents the point of view of a publisher's book production department when accepting author-supplied T<sub>E</sub>X manuscripts. Topics covered include tips for authors and publishers, L<sub>A</sub>T<sub>E</sub>X vs. T<sub>E</sub>X vs. A<sub>M</sub>S-T<sub>E</sub>X as publishing tools, creation of house macros, and use of freelance services.

## Introduction

**History.** The use of T<sub>E</sub>X at Springer-Verlag New York began sometime in the early 1980s with the acceptance of "camera-ready" copy from authors. Unlike our parent company in Germany, no macros were developed and no guidelines exclusive to T<sub>E</sub>X were written. Authors had only our general camera-ready guidelines to guide them and those guidelines were developed for the lowest common denominator, namely, typewriter copy.

By the middle to late 1980s, author pressure to create their books using T<sub>E</sub>X increased to the point that L<sub>A</sub>T<sub>E</sub>X macros were developed using a consultant. This enabled us to do two things: accept author-supplied copy on a standardized basis and employ freelance T<sub>E</sub>X typesetters to set copy from paper manuscript. At this point output was from 300-dpi laser printers and the quality difference was clearly visible between books done in T<sub>E</sub>X and our conventional mathematics typesetters.

During this time no T<sub>E</sub>X expertise was being developed in the Book Production Department. There were two reasons for this lack of expertise: (1) at that time people who were employed in book production departments had little or no computer skills or experience; and (2) little or no equipment was available to gain that experience.

In the last two to three years the situation has changed dramatically. The tremendous increase in author use of T<sub>E</sub>X, improved output capabilities, and growing availability and experience of T<sub>E</sub>X suppliers has made T<sub>E</sub>X not only a viable part of any

mathematical or physical sciences production department but a star.

## Production Outlook

**Costs.** One of the main functions of a production department is to spend money. The success of any department is determined by not spending any more than is necessary to produce a quality product. T<sub>E</sub>X has the capability of producing a quality product at a low cost. However, several factors can negate this cost-saving possibility: (1) author fees for providing hard copy or electronic files; (2) high output costs; (3) high freelance costs for typesetting or reformatting; (4) overhead costs when authors do not use macros or have problems in the final stages of production.

**Quality.** The word quality can evoke different visions from authors and production editors. Both agree it means the lack of typographical errors and errors in fact. At that point, however, the visions may diverge.

The area of greatest potential conflict is design. Authors with the power of computer technology and all the fonts that a few hundred dollars can provide, want to express some creativity not only in the words that appear on the page but in the appearance of the words themselves. Book design, however, is both an art and a science. Production editors with a decade or more of experience hesitate to change department specifications to fit a particular book, but authors who have read a few dozen books want to

repeat catchy elements that they have seen in several of them. Book design and typograghy should be left to professionals. That does not mean that some authors cannot produce quality design work, but the continued submissions of "ransom note" designs should lead departments to discourage this practice.

Error prevention is an important quality consideration. Is there anything more distracting than reading a book with the appearance of a high-quality product and finding typos and other annoying errors? One service that conventional math typesetters provide is professional proofreading. This can be lost when an author or a small freelance service is setting the book. We try to recover this lost service by copyediting in the page proof stage. The copy editor performs both copyediting and proofreading functions and also serves as a design reviewer. This step has saved authors from embarrassing errors in books they thought were final and clean.

## Hardware

TeX coding is the same on all platforms, but user interfaces can vary widely. The variations of interfaces cause differences in flexibility, disk management, learning curves, and of course speed. A large part of deciding on which platform to run TeX can be derived by available equipment and budget. There are high-, middle-, and low-end TeX hardware setups.

Springer uses what could be considered a "middle-end" hardware setup. We currently run TeX in a combined DOS-Macintosh environment. Our setup grew out of a basic office computing scenario. Production editors used PCs for word processing, spreadsheets, and database work. Macintoshes were recently brought in to bolster our art program and increase our desktop publishing potential. When we decided to bring desktop publishing (including TeX) into the department, we chose to upgrade our current configuration rather than start from scratch. A few more computers were added. These were equipped with more memory, faster processors, and larger storage devices. Book-length TeX files are huge. We often find these files (particularly dvi files and output files) in competition with other applications for storing files in our hard drives. We even added a local area network, a pair of modems, and a scanner. In this way, TeX files can be edited and coded at one station and massaged at another. Proofs can be generated on any of our bitmap and PostScript laser printers.

More important, this configuration allows us considerable flexibility in accepting author-

generated files. We can take files from DOS, Macintosh, and UNIX environments, either via disk for large files or telecommunications for short ones. Authors working in the UNIX environment are asked to download their large TeX files onto DOS- or Macintosh-formatted disks. Authors working on PCs or Macintoshes send in their files as is.

**DOS.** TeX runs well in both the PC and Mac environments. The DOS version utilizes separate programs that can be mixed and matched. A variety of text editors, TeX engines, dvi previewers, and printer drivers can be used. Each can be obtained at reasonable prices. This gives great flexibility for users. An added benefit is that a user's "tried and true" text editor can be used with maximum efficiency. A major drawback is the fact that each program must be run separately in order to create hard copy. Each program requires a given amount of startup time. Adding startup times for the four necessary programs (editor, TeX engine, previewer, and print driver) easily creates a significant amount of time merely waiting for the DOS version to load.

**Macintosh.** The Macintosh version offers a fully integrated interface. The text editor, engine, previewer, and driver are all built-in. Unlike the DOS version, only one program is needed to create final copy. Some consider the Macintosh version easier to learn. However, the higher prices of Macintosh hardware and software make use of these machines in large quantities more difficult in a book publishing atmosphere.

Our mixed environment allows us to leave all machines application non-specific. We have not dedicated any of our machines to only running TeX. All of our machines can be used for general word processing, spreadsheets, database work, and other desktop publishing applications, as well as for TeX. The network allows us to keep work constantly moving. Text files, macros, and style files can easily be accessed throughout the network. Production editors working on a project are not anchored to only one machine, thereby increasing productivity. Some might say that TeX does not perform at its absolute best in this environment, but Springer is not in the business of becoming a composition house. As a publisher, we set out to create a working environment in which we could accept TeX files, edit last minute changes, and obtain final copy. Raw manuscript and major editing projects are routed out of house.

**UNIX.** An alternate, higher-end setup would utilize UNIX. Workstations and software could be purchased at a premium price. This high price tag

would allow for increased speed, true multi-tasking, and  $\text{\TeX}$  working faster. UNIX systems are mainly used by programmers and as yet have made little headway into the general business marketplace.

### **$\text{\LaTeX}$ vs. $\text{\TeX}$ vs. $\text{\AMS-\TeX}$**

The three most common varieties of  $\text{\TeX}$  are  $\text{\LaTeX}$ , plain  $\text{\TeX}$ , and  $\text{\AMS-\TeX}$ . Each has its strengths and weaknesses.

**$\text{\LaTeX}$ .** From a publisher's point of view,  $\text{\LaTeX}$  is the version of choice. It is more structured than the others. It was created by Leslie Lamport for authors to create books and other documents. The macros are very comprehensive. Authors using  $\text{\LaTeX}$  almost always use the well-defined macros available in  $\text{\LaTeX}$ , rather than creating new ones. This reliance on the  $\text{\LaTeX}$  coding schemes makes it very easy for publishers to apply their own macros. The publisher simply alters the pre-existing set of  $\text{\LaTeX}$  macros. The author need not learn any new commands. As far as the author is concerned, he is using the standard  $\text{\LaTeX}$  codes.

As mentioned above, the  $\text{\LaTeX}$  coding scheme creates a very structured design that relies more on proper layout technique than visual appeal. Good layout is often the primary ingredient for creating a good-looking book.

**Plain  $\text{\TeX}$ .** On the other hand, plain  $\text{\TeX}$  and  $\text{\AMS-\TeX}$  are far more flexible. Using plain  $\text{\TeX}$ , the author has near total control over the look of the book.  $\text{\TeX}$  starts from scratch. There are few readily available layout macros built into the system. Authors must create their own macros or define the document line by line. Many authors prefer this flexibility as it keeps them from feeling trapped in a generic design. Such an attitude can cause problems for the publisher when trying to produce the book. Few authors are trained designers, and many times what looks nice on single sheets translates poorly to a bound book.

It is possible for publishers to supply useful macros for plain  $\text{\TeX}$ . As there are no initial overall layout commands in plain  $\text{\TeX}$ , these macros would introduce an entirely new set of commands for the author. Therefore, authors should use these commands as they are creating their chapters. This is different from  $\text{\LaTeX}$  where an author is familiar with the basic  $\text{\LaTeX}$  commands and therefore is readily familiar with the macro commands.

A detailed set of instructions *must* accompany the macros, as the authors must be taught the macro set and how to use it. Designing plain  $\text{\TeX}$  macros brings up the problems of creating macros from

scratch. This topic will be detailed in the section on house macros.

Another problem can also arise from author-created plain  $\text{\TeX}$  macros—a seemingly innocuous author-created macro could unknowingly conflict with the publisher's macros, making a tremendous mess when run through  $\text{\TeX}$ .

**$\text{\AMS-\TeX}$ .**  $\text{\AMS-\TeX}$ 's greatest asset is its easy accessibility to the AMS fonts. This extra set of fonts allows mathematicians to utilize a number of special mathematical characters. These fonts can be accessed by  $\text{\LaTeX}$  and plain  $\text{\TeX}$ , but not as easily.  $\text{\AMS-\TeX}$  has some built-in structure. The  $\text{AMSPPT.STY}$  is a layout format designed for typesetting a mathematical paper. Books have different demands. Publisher macros can be used to overlay the  $\text{AMSPPT.STY}$ , but the coding set available is not as comprehensive as the one available for  $\text{\LaTeX}$ .

Using the AMS fonts can be a problem if not handled carefully. Some high-resolution output services do not have these fonts. The fonts can be obtained, but often this leads to schedule delays and font "bugs," such as character crashes and font priority problems.

In the end, plain  $\text{\TeX}$  and  $\text{\AMS-\TeX}$  can be used to create a high-quality book, but they will require more effort on the part of the publisher and the author to do so.  $\text{\LaTeX}$  was designed to make books, and with a little tweaking provided by publisher-designed macros, high-quality products are ensured.

### **Developing House Macros**

The development and use of house macros is the most critical area for success for a book production department using  $\text{\TeX}$ . House macros are the meeting place for production departments and authors.

Macros should have three qualities: (1) they should be easy to use and concise; (2) they should be flexible enough to allow the author to express himself even when confined to a house style; and (3) they should be accessible. If these qualities are not there, authors will not be inclined to use the publisher's macros, and more work for the production department will result.

Ease of use depends on the accompanying documentation in the form of an instruction file as well as the generous use of comment lines within the macro itself. Flexibility is the result of the house macro operating on the format aspects of the copy only.

Macros should be available for both single-author and edited volumes.

The easiest way to get started with house macros is to hire a  $\text{\TeX}$  specialist as a consultant.

There are several advantages to this: (1) the startup speed of your TeX program will be that much faster; (2) you do not use valuable in-house time on something your department is not suited for; (3) you get professional expertise and a set of macros tailored to your house style; and (4) you get follow-up down the road when the consultant works as a troubleshooter for you. The disadvantages are: (1) the cost of hiring the consultant; (2) new macros almost always have bugs that need to be worked out; (3) if you have little or no TeX knowledge and the consultant is not working closely with you, you will have a set of macros that you do not understand and that you cannot explain to your authors.

Another method for obtaining house macros is to adapt an already existing macro for your department's use. The advantages here are: (1) the cost is low because you are not paying a consultant and you are not spending a lot of in-house time; (2) the macro is generally bug-free; (3) this forces you to learn more about TeX and will enable you to adapt to unique situations when they arise.

At Springer-Verlag New York we undertook a combination of these two approaches. First, when we had no TeX expertise in-house, a consultant was hired to develop a L<sup>A</sup>T<sub>E</sub>X macro for us. This enabled us to get the ball rolling on TeX and have a "welcome mat" available for our authors. We then worked to get our TeX expertise improved internally. It did not take more than a few months of accepting L<sup>A</sup>T<sub>E</sub>X books for us to feel comfortable both with L<sup>A</sup>T<sub>E</sub>X and with the macros that were developed for us.

We also began picking up experience with plain TeX. We borrowed a suitable plain TeX macro package from our Heidelberg production department and adapted it for our own needs. We left 95% of the macro package untouched, therefore it was free of bugs.

After two years of this kind of experience, we are now at the level where we can begin thinking of developing our own macros. We also feel comfortable helping our authors with problems on their macros.

## TeX Suppliers

The use of service suppliers for TeX is a key to the success of TeX in the production environment. There are three types of services that can be provided out-of-house: keyboarding, reformatting, and full service.

**Keyboarding.** The availability of TeX keyboarding services has grown in the last few years. Because

a lot of overhead is not needed (all that would be required would be a microcomputer, software, and printer), services that provide only keyboarding can be quite economical for math typesetting. Springer was already using keyboarding services to provide camera-ready copy. We are now moving away from that for two reasons: (1) the keyboarding service was not providing true full service; and (2) most authors for math books are already providing TeX files for our books. Our keyboarding services are now moving to the function of inserting author and copyeditor corrections to an already existing TeX file.

**Reformatting.** Another service necessary for out-of-house work is reformatting of existing TeX files to house specifications. Some of this work might include dimension and font changes, figure placement or spacing, running head preparation, and insertion of house macros.

Whether Production can use these services is decided by cost. A comparison must be made between the costs of this being done out-of-house, versus in-house (overhead), or just using the TeX output as a well-prepared manuscript and typesetting through non-TeX sources.

**Full Service.** By full service we mean accepting a paper manuscript or a TeX file on diskette and providing keyboarding, proofreading, formatting, macro writing, illustration work, and output. This is essentially providing the same services as any conventional typesetter, just using TeX to get it done. For the Production Department there is no distinction between these services and conventional typesetters. They should have to compete on a cost, quality, and schedule basis.

## Fonts, Figures, and Final Output

There are currently two tracks used for outputting TeX documents: bitmap and PostScript.

**Bitmap.** Following the bitmap track, one primarily uses TeX's standard Computer Modern typefaces and has them output at high resolution. Non-TeX coded figures are stripped in by hand onto the final pages. This is done by the conventional method of cut-and-paste using wax, a blade, and a light table. The bitmap track provides a high-quality final product at low cost and easy maintenance, but is very limited in flexibility. Last-minute changes are difficult to make as they often mean having to restrip art.

**PostScript.** The PostScript track offers a much more flexible environment. A user can choose from

a huge library of PostScript fonts at varying resolutions. Figures can be placed electronically. Figures can be scanned into *encapsulated* PostScript files or drawn using various computerized art programs, then stored as an encapsulated PostScript file. These files can then be imported using *TeX*'s \special command.

PostScript is the mainstream font language of the desktop publishing community. If a user creates PostScript output files rather than *dvi* files, he can take advantage of the thousands of PostScript output vendors, both locally and internationally. This huge output service community keeps the market volatile. Prices depend on resolution required, turnaround, and volume. Unfortunately, most PostScript vendors are not *TeX*-aware. If a user can only create *dvi* files, they have greatly reduced the number of available output services. *TeX*'s *dvi* file concept is foreign to most service bureaus. Being able to explain the *dvi* file concept and judge a bureau's technical expertise requires more in-house knowledge on the part of the publisher.

Springer-Verlag New York is currently following the bitmap track though we are progressively moving toward the PostScript track. However, our current plan is to not move completely out of the bitmap track, but look at output on a case-by-case basis. Using the standard *TeX* *dvi* files to obtain high resolution output is still the easiest and most cost-efficient means for obtaining camera copy.

## Author Submissions

There are many ways for authors who submit *TeX* files to help a production department handle their files in a quick and economical way. Authors should provide source, macro, and *dvi* files as well as hard copy to the publisher. They should check with the publisher to determine the disk, tape, and other formats required for submission.

Naming of files is important. Authors writing a book a year can title their files *ch1.tex*, etc., but this will force the production department to rename files so they are not lost among all the other *ch1.tex*'s. This applies to macro files as well. Naming your files with the part of the first author's last name should be the standard (e.g., *Spr1.tex*). Do not name files *book.tex*, *preface.tex*, *macro.tex*, etc.

Authors should use the publisher's macros. If they want to modify those macros for their book, they should speak with the production department.

## Staff Training

Perhaps the best route for training staff in *TeX* is to send them to a *TeX* Users Group training course and then apply that knowledge in-house on numerous projects. In a few short months, a book production department often sees a wide variety of *TeX* books. This atmosphere provides a rich training ground. Students about to learn *TeX* should have already acquired strong word processing experience, especially in *search and replace* techniques (macro creation is an added plus). Knowledge of file manipulation is also strongly recommended. *TeX* creates an enormous number of files. These files must be kept in very specific locations. Simply knowing the difference between subdirectories on a PC or folders on a Mac is without question invaluable.

Springer has also discovered that a varied desktop publishing environment also helps build expertise in *TeX*. The ability to conduct staff brainstorming sessions in *TeX* makes a production department an arena for quick learning.

## Secondary Usage

The day is very near (if not here already) when the paper product will not be the only form in which a book is published. Standardization of electronic files will become a requirement in the coming years. Authors now have the ability with *TeX* to e-mail files to colleagues for almost instantaneous interaction over long distances.

SGML is the ultimate standard and may be the goal for production departments when considering the handling of electronic products. This stands *TeX* in good stead for many years to come. First, it is ASCII-based and easily lends itself to translation to SGML forms. Second, it already is something of a standard in math and physical science departments around the world, giving *TeX* a great advantage over the proprietary systems of most mathematics typesetters.

## Conclusion

Any publisher in the science and technical area, particularly in the mathematical and physical sciences, is going to have to deal with *TeX* in order to produce cost-competitive books in a high-quality way. Knowledge must be gained in production departments to deal with *TeX* files, and author-publisher interaction in this area must be supported. Production departments that treat *TeX* as a black box will not be gaining everything that they can from *TeX*.

# dvi and EPS: The Ideal Author-to-Publisher Interface?

Berthold K.P. Horn

MIT AI Laboratory, 545 Technology Square, Cambridge, MA 01741 USA

Internet: [bkph@ai.mit.edu](mailto:bkph@ai.mit.edu)

## Abstract

dvi files specifying text and document format, along with EPS (Encapsulated PostScript) files for included figures, are rapidly becoming the *de facto* standard for interchange of machine-readable manuscripts in technical publishing. While dvi file format and EPS file format are standardized, the glue holding them together, namely the \special macro, is not. This is presently the weak link in the chain. Unfortunately, in the absence of an officially sanctioned standard, every publishing organization is developing its own standard, often inelegant and inextensible. Electronic publishing has arrived. Publishers in specialized technical areas are using machine-readable material now. Their needs for standardization have become critical.

## Introduction

Device-independent (dvi) files specifying text and document format, along with Encapsulated PostScript (EPS) files for included figures, are rapidly becoming the *de facto* standard for interchange of machine-readable manuscripts in technical publishing.

The advantage of dvi files over raw T<sub>E</sub>X files is that there is no need for the publisher or service bureau to bring up the special version of T<sub>E</sub>X used by the author, nor does the publisher have to deal with the author's macro packages. dvi files are supremely standardized, portable and compact.

The advantage of dvi files over PostScript (PS) files produced by present-day dvi-to-PS converters is that dvi files are resolution-independent, while PostScript files containing bitmapped fonts are not. As long as dvi-to-PS converters continue to use bitmapped fonts, they will have to be run over the dvi file again and again, each time an output device with different resolution is to be used.

While dvi file format and EPS file format are standardized, the glue holding them together, namely the \special macro, is not. This is presently the weak link in the chain. Unfortunately, in the absence of an officially sanctioned standard, every publishing organization is developing its own standard, often inelegant and inextensible.

The only viable alternative to the combination of dvi and EPS files is the resolution-independent PS file. Resolution-independent PS files containing both text and illustrations are possible now that

high-quality outline font programs are available for Computer Modern.

One advantage of resolution-independent PS files over dvi files is that they contain only ASCII characters and so can be more conveniently stored and transmitted. Perhaps more significant is the fact that resolution-independent PS files can be sent to a service bureau that is not knowledgeable about T<sub>E</sub>X and dvi and does not have access to high-resolution bitmapped fonts. This lowers costs considerably and gives the editor or author complete control over the final appearance of the work.

## The Best Medium of Interchange?

We probably would all agree that when writing on a technical subject, particularly one requiring the use of mathematical formulae, an author these days finds few viable alternatives to the use of T<sub>E</sub>X for preparing papers and books. In the past, the author's manuscript, after review and revision, was typeset, with the author required to proofread the result, which quite often was less pleasing than the original T<sub>E</sub>X output submitted!

This whole process is expensive, slow, frustrating, and error-prone. It is, of course, being displaced by the obvious alternative. But so far this transition has been slow and painful. There are a number of critical areas that need urgent attention if the change is to progress more smoothly.

First of all, what is the best medium of interchange between author and publisher? Should

it be (a) *T<sub>E</sub>X* source, (b) *dvi* files, or (c) PostScript code produced from *dvi* files? I will argue that at the present state of development the *dvi* file is the best of the three alternatives. The reason is that *dvi* files are standardized, portable, and compact. (And unlike some other ‘standards’ the format of *dvi* files *really* has not changed for many years.)

**T<sub>E</sub>X source and macro files.** If the author supplies *T<sub>E</sub>X* source, the publisher or typesetting service bureau needs to be able to run the dialect of *T<sub>E</sub>X* used by the author, and also have access to the macro packages used. This may involve moving a complex web of interrelated files. More seriously, it requires considerable investment in computer hardware, software, and a level of sophistication that is *not* required if only *dvi* files are being manipulated.

Another problem is that the publisher may have stale versions of some of the macro files. One way to make the use of *T<sub>E</sub>X* source slightly more bearable, and to circumvent the stale macro file problem, is to create a program, called *TeXExpand* perhaps, that creates a single (large) file by (recursively) inserting files called for in the original *T<sub>E</sub>X* source file.

**PostScript code.** PostScript code generated from *dvi* files in the past was not resolution-independent, since *dvi*-to-PS converters used bitmapped fonts. This meant that the publisher had to tell the author in advance what device the text would be typeset on, and the author had to build the (large) bitmapped font files required for that device.

Typesetting could not proceed from the same file used by the author to produce draft output for review. The publisher did not have the ability to later alter the choice of output device, since the resolution was frozen in the files.

**dvi files.** The above clearly suggest that *T<sub>E</sub>X* source and PostScript output are less satisfactory than *dvi* files. To many people the idea that the *dvi* file is the best medium of interchange is so alien that, even after being told several times to send *dvi* files, they continue to submit PostScript files; and when reminded not to do this, they will send *T<sub>E</sub>X* source files along with a web of macro files!

The only minor drawback of *dvi* files is that they are binary, requiring care in transmission.

## What about Illustrations?

Next we come to the question of illustrations. Presently the most satisfactory method here appears to be the use of encapsulated PostScript files. Properly constructed—that is, conforming—EPS

files can be resolution-independent and print well on any PostScript image-setting device. Some of the alternatives are less satisfactory, although they have their uses in specialized situations, e.g.:

- a. *L<sub>A</sub>T<sub>E</sub>X* *line* and *circle* fonts permit construction of certain kinds of simple figures;
- b. *P<sub>I</sub>C<sub>T</sub>E<sub>X</sub>* can generate graphs and figures of limited complexity; and
- c. *METAFONT* can generate bitmap images, although these are not resolution-independent, and will look ‘pixelated’ when printed on a high-resolution device.

In most cases then, the combination of *dvi* and EPS files appears to be the best combination for transfer of material from the author to the publisher.

Indeed, *dvi* files specifying text and document format, along with EPS files for included figures, are rapidly becoming the *de facto* standard for interchange of machine-readable manuscripts in technical publishing.

## *dvi* Files are *not* Device Independent

The only problem with this rosy picture is that *dvi* files are not truly device independent! Yes, unfortunately there are two areas in which the extensions provided for by *T<sub>E</sub>X* lead to difficulties. And these extensions are the very ones that we can no longer imagine living without. They are:

- a. Inserting illustrations using *\special*; and
- b. Using fonts other than those in the Computer Modern family.

## What's so Special about *\special*?

The problem with use of *\special* for figure insertion is the more complex of the two problems, but also the one more urgently in need of a solution. In the absence of an officially sanctioned standard, every publishing organization using *T<sub>E</sub>X* or *dvi* files is developing its own *de facto* standard, sometimes inelegant or inextensible.

A major stumbling block to completion of the transition to electronic publishing is that every *dvi* processing program supports a different convention for usage of *\special*. This means that every job is a custom job. Instead of a smooth operation involving only the transfer of the author's *dvi* and EPS files, a serious programming effort is often required to deal with yet another way of using *\special*.

While *\special* is the open door to extensions of *T<sub>E</sub>X* usage, we need concern ourselves here

only with the use of `\special` for figure insertion. Arguments over uses of `\special` for other purposes should not drown out discussion of the urgent need for a simple standard for figure insertion.

No one can anticipate all the possible uses for this powerful extension of the *TeX* language, nor is it likely that the community can soon agree on the details of how such extensions are to be implemented. But this should not stand in the way of satisfying what has now become an urgent need: a standard way of using `\special` to include figures in text.

When the discussion of standards for dvi drivers first started, there was little urgency, since the routine need for these capabilities had not yet arisen outside a small number of research laboratories. Progress in this field has been rapid, however, outpacing deliberations of the standard committees, with journals rapidly switching to machine-readable material. Similarly, books are now routinely produced from *TeX* output.

This represents one of two major obstacles to seamless electronic publishing. Therefore,

- The time for publication of a simple standard for figure insertion in papers and books is *now*.

**Simple requirements.** What is required is in fact really quite straightforward. All that is usually needed is a means for inserting a figure, possibly scaled, at the desired position in *TeX*. Sometimes it is also useful to be able to shift and perhaps rotate and clip the figure. Informal statistics show that 80% of the time simple figure insertion is enough, while scaling is also called for in perhaps 20% of the cases. Shifting, rotating, and clipping are almost never used, but should perhaps be provided—just for generality's sake.

More important, no use seems to be made of the ability to insert verbatim PostScript commands, to call on PostScript functions native to a particular dvi processing program, or to produce overlays. While these transformations represent interesting and powerful extensions, they apparently are not vital to the production of even the most sophisticated texts.

There may be several reasons for the limited use authors presently make of the more complex figure manipulations:

- a. Apparently even the most sophisticated textbooks can be produced using little more than simple figure insertion.
- b. It is relatively easy to modify a file that obeys the EPS structuring convention to achieve the desired graphical transformation.

- c. Authors know that exploiting esoteric features of particular dvi processors will reduce the portability of their document and consequently restrict themselves to the simplest operations that will accomplish their objective.

Lack of standardization of usage of `\special` for figure insertion is the main obstacle to seamless electronic publishing using *TeX*. A simple standard is urgently required.

**Existing schemes.** For inspiration one might consider some of the existing schemes:

- a. The use of `\special` in UNIX's DVI2PS is simple, and provides most of the listed features. An example:

```
\special{psfile=figure.eps
        hscale=0.66 vscale=0.66}
```

- b. The use of `\special` in Blue Sky Research's *Textures* is also satisfactory, although it does not provide all of the features indicated (but in turn provides some others). An example:

```
\special{illustration figure.eps
        scaled 667}
```

- c. The proposed use of `\special` in Nelson Beebe's next release of DVIALW has many desirable features (although it is perhaps more complex than needed). For example:

```
\special{language=PS include=figure.eps}
```

It should be possible to use `\special` for figure insertion without reference to internal procedures of a particular dvi-to-PS converter or inclusion of verbatim PostScript code.

It should be clear in any case that a standard syntax for figure insertion using `\special` should be established as soon as possible.

## Font-Naming Woes

The other device-dependent aspect of dvi files is the naming of non-Computer Modern fonts. This is the easier of the two problems to analyze—and to fix.<sup>1</sup>

For Computer Modern there exists a standardized way of relating the font names used in *TeX* to the files containing font metric information and the files containing the actual outlines or bitmaps of that font.

<sup>1</sup> The reason the discussion of the font naming problem covers more paper here than discussion of the more serious problem of standardization of `\special` for figure insertion is precisely that it is the simpler of the two issues.

We do not usually waste much time worrying about this, but there needs to be a mapping between three entities: (a) the name used to refer to a font in the *T<sub>E</sub>X* source document, (b) the name of the *T<sub>E</sub>X* font metric (*tfm*) file for that font (which *T<sub>E</sub>X* needs to do its job), and (c) the name of a font program file (or a font program in the printer) that actually draws the characters (which the *dvi* processing program needs to know about).

Unfortunately, there is no general agreement yet on how to build such a mapping for fonts other than Computer Modern. The problem would be slightly simpler if it were not for the fact that the name used to refer to a font in *T<sub>E</sub>X* used to be constrained to be no more than 6 characters long—and is in any case constrained to no more than 8 characters by some operating systems such as MS-DOS. What is done now—as a stop gap measure—is for *dvi* processing programs to accept an auxiliary file that contains the mapping. This file must be supplied by the author or constructed by the publisher after obtaining the required information from the author.

One reason the font-naming problem is becoming more of an issue is that many publishers are urging authors to be more ecumenical about font selection. So far, such pressures have encountered strong resistance because of the sparsity of satisfactory non-CM fonts for typesetting mathematical formulae. But there is now at least one alternative: Bigelow and Holmes' LucidaMath fonts published by Adobe.

**Lack of portability.** This lack of standardization has proved to be a source of frustration when *dvi* files are ported from one computer system to another, as is common when publishing journal articles and books from author-supplied material. As it stands now, each project requires customization, compelling the typesetting service bureau to set up yet another new font-name translation table. Perhaps more seriously, without a uniform naming convention, it may happen that the *dvi* processing program and *T<sub>E</sub>X* have conflicting notions about what fonts are being referred to—with disastrous consequences.

The above represents the other major obstacle to seamless electronic publishing. Therefore,

- A standard naming convention for fonts other than those in the Computer Modern family should be established as soon as possible.

This is particularly important for the existing collection of fonts in Adobe Type 1 format. This collection is both popular and very large. Thirty

vendors supply over 7000 fonts in this format (at the time of writing), with 1300 in the Adobe Font Library alone. Here an unaesthetic standard is better than no standard at all—or a standard that is not extensible enough to deal with the continuing flood of typefaces being converted into this format.

One solution would be to establish some permanent organization to invent abbreviations or at least act as a clearing house for proposed abbreviations of font names. This does not seem practical, since it is unlikely that such an organization could deal in a timely fashion with the rapid growth in the collection of fonts in this format. Consequently,

- One should use established font-naming schemes whenever possible.

This will reduce confusion and avoid the need for a central registry of abbreviations. Adobe, for example, has already found it necessary to invent 6-character abbreviations for its fonts—it seems inefficient not to use these.<sup>2</sup> This in fact will take care of a significant part of the font-naming problem, since presently the most commonly called for non-CM fonts are Adobe Type 1 fonts.

**Remapping of character code assignments.** Unfortunately, the above isn't the full story. Each font has its own mapping between the numeric character codes (typically 0–255) and character glyphs. There are nine different standard mappings used by Computer Modern fonts: roman (e.g., *cmt10*), text italic (e.g., *cmti10*), typewriter (e.g., *cmtt10*), typewriter italic (e.g., *cmtti10*), small caps (e.g., *cmcsc10*), *T<sub>E</sub>X* ASCII (e.g., *cmtex10*), math italic (e.g., *cmmi10*), math symbol (e.g., *cmsy10*), and math extended (e.g., *cmex10*).

A non-CM font can be used with the encoding it came with, or can be remapped to one of the above standard encodings. It must be possible to distinguish between *tfm* files for the original font and the remapped font. The easiest way to do this is to use different, but related, file names for the two versions. One simple scheme is the following:

- The file name of the *tfm* file for a remapped font has an 'x' appended.

This doesn't completely solve the problem, since it doesn't specify *which* remapping was chosen. Unfortunately, the *tfm* file format does not provide for an encoding vector mapping numeric character

---

<sup>2</sup> Some of Adobe's font downloaders happen to limit the font-name part of the file name to six characters, which conforms exactly to the old restriction in *T<sub>E</sub>X*.

codes to character names, only an *optional* field that may contain the name of a remapping (and only the nine names mentioned above are in any way considered standard).

Fortunately, the need for remapping fonts is greatly reduced by the advent of TeX 3.0, which can deal with 8-bit character codes.

## Resolution-Independent PostScript

The only viable alternative to the combination of dvi and EPS files is the resolution-independent PS file. Resolution-independent PS files containing both text and illustrations are possible now that high-quality outline fonts are available for Computer Modern.

**An aside.** Some readers may have low expectations for the quality of rendering using outline fonts, perhaps having seen the results of some early experiments. Properly hinted Type 1 fonts, however, are compact, support across-job font-caching, and most important, produce beautiful characters. Type 3 outline fonts, used in some early experiments, suffered from the ‘dot-growth’ phenomenon inherent in use of the PostScript *fill* operator. Furthermore, *unhinted* Type 1 fonts do not render well on low resolution devices such as computer display monitors.<sup>3</sup>

To return to the topic at hand, note that resolution-independent PS files derived from dvi and EPS files:

- a. should not make any assumptions about the output device resolution;
- b. should not rescale or round coordinates given in dvi files; and
- c. should not refer to bitmapped fonts.

One advantage of resolution-independent PS files over dvi files is that they contain only ASCII characters and so can be more conveniently stored and transmitted. (Extra work is required to safely transport binary files across networks or even serial lines connecting disparate computer systems.) There is no need to redo the conversion from dvi to

---

<sup>3</sup> Also, a particular character’s shape may be described in many different ways by using lines and Bézier curves. Some such description may contain many more elements than really necessary, and may not obey the strict rules specified in the Type 1 standard. Rendering using such an outline is likely not to be as fast or as clean as that of a properly constructed outline.

PS form when a printer or image-setter of different resolution is used.

Perhaps more significant is the fact that resolution-independent PS files can be sent to a service bureau that is not knowledgeable about TeX or dvi files, and does not have access to high-resolution bitmap fonts. This lowers costs considerably and gives the editor or author complete control over the final appearance of the work.

## Summary

dvi and EPS files are the preferred medium of interchange of material between author and publisher.

Electronic publishing has arrived—although it is not quite seamless yet. Publishers in specialized technical areas are using machine-readable material now. Their needs for standardization have become critical.

One of the areas in need of attention is that of usage of \special for inclusion of illustrations:

- A standard syntax for figure insertion using \special should be established as soon as possible.

This should not close the door on future, as yet unanticipated, uses of \special. All that is needed now is a simple syntax for insertion of illustrations. There is serious danger that in the absence of any guidance *ad hoc* standards will come into widespread use that are neither elegant nor extensible. The window of opportunity for influencing developments in this area is open now, but will not remain open indefinitely.

The other problem area is that of naming conventions for fonts other than Computer Modern:

- A standard naming convention for fonts other than those in the Computer Modern family should be established as soon as possible.

Finally, note that there is an alternative to the use of dvi and EPS files, namely the resolution-independent PS file. As a parting thought, consider the following table of estimated costs:

- \$30–40 per page for traditional typesetting;
- \$9–10 per page for service bureau work from TeX source; and
- \$2–3 per page to print resolution-independent PostScript.

There is a clearly an incentive to consider seamless electronic publishing. And there is clearly an even greater incentive to consider resolution-independent PostScript.

# Producing a Book using $\text{\TeX}$ : How the Process Works

Neil A. Weiss

Department of Mathematics, Arizona State University, Tempe, AZ 85287

602-965-3951; FAX: 602-965-8119

## Abstract

The steps required to carry out the production of a book using  $\text{\TeX}$  will surely vary somewhat depending on the author and publisher. However, many aspects will be similar. In this paper, we will trace the production of a two-color introductory statistics book that was typeset using  $\text{\TeX}$  and some PostScript. Since the book was produced in earlier editions using traditional methods, we will also discuss some of the advantages and disadvantages of producing a book using  $\text{\TeX}$ .

## Introduction

As any author will attest, writing a book is a difficult, time-consuming, and often frustrating process. But the completion of the final manuscript is only the beginning! After that, the book must go through a production cycle in which a design is composed; the art is drawn; the text is copy edited, typeset, and dummed; and the book is manufactured.

In the traditional method of typesetting, a compositor is given a copy-edited manuscript to typeset. The next thing that the author sees is the galley proofs, long pages of typeset material with no art in place and no multiple columning. Depending on the quality of the compositor, the galleys can vary from being an excellent rendition of the manuscript to an absolute disaster.

After publishing several books using traditional methods, my sentiments were very close indeed to those expressed by Robert Adams [1990]: "...It was my second book done by the *old method*, and I resolved at the time never to write another book!"

Then in 1986, as fate would have it, I saw a magnified page out of *The  $\text{\TeX}$ book* (page 77, I think) that was posted on a bulletin board. Upon investigation, I discovered that there was this wonderful program called  $\text{\TeX}$  that could be used to typeset technical manuscripts with amazing typographical precision—and it could be done on a personal computer!

My wife Carol and I decided to give  $\text{\TeX}$  a try by first using it to produce some supplements to a textbook that had just been published. With the help of Professor Thomas Sherman and Carol's

incredible insight, we published the  $\text{\TeX}$ -produced supplements. I was delighted with the look of the supplements and with the control that I maintained over their production. It was then I decided that, if at all possible, my next textbook would be done using  $\text{\TeX}$  and that it would be typeset by Carol and me.

Thanks to the faith that our publisher had in us, we were given the opportunity to use  $\text{\TeX}$  to typeset the third edition of the book, *Introductory Statistics* [1991], by Matt Hassett and me. This paper traces the steps that were taken to produce the book.

## The Design Stage

The first step in the production of the book was the design stage. After preliminary discussions with us (Carol and me) and the publisher, the designer constructed the design specifications, called the "specs," based on the previous edition of the book. If the book had been a first edition, the designer would have used the manuscript as the basis.

The purpose of the specs is to provide a complete description of every aspect of the design. This includes the trim size, margins, color separation, fonts, and detailed instructions for all elements—chapter openers, first- and second-level heads, definitions, tables, figures, etc. The designer also provides a coding for the elements that is used by the copy editor and the macro writer. For instance, the following portion of the specs for the introductory statistics book gives the details for setting the examples and their solutions:

```
EX word EXAMPLE and its Arabic double
number cmssi 12/12 fl right in margin
col. 15pts bb above to a 1pt rule x 7.
Print head and rule color. Base align
with ET.
```

```
ET example title
cmssi 12/12 C/1c fl left x 30, rr.
33pts bb above, 18pts bb below to
example text (basal). 24pts bb below
to SOL; or (if no solution), use
square per end of SOL and 9pts bb
below to end 1pt rule x 30 (color)
which normally follows SOL. Min 30pts
bb below this rule to basal.
```

```
SOL solution
cmssi 9/12 caps fl right x margin col.
Base align with first line of solution
text (basal). Print color. 24pts bb
above. Set a solid 6pt square fl r x 30,
base aligned with last line of SOL.
Clear 1em to left of square. Print color.
9pts # below to 1pt rule x 30 (color).
Min 30pts bb below rule to basal.
```

Let me interpret the first two of the three design specifications displayed above. EX is the code used for "example." In this case, the word EXAMPLE is to be set in all caps followed by its Arabic double number (e.g., EXAMPLE 8.12). The font to be used is cmssi12, twelve-point computer modern sans serif italic. Also, the word EXAMPLE is to be positioned flush right in the margin column (which is 7 picas wide) with a 1-point-high and 7-pica-wide rule over it, and a 15-point baselineskip. The word EXAMPLE, its double number, and the rule are to be printed in color, and the word is to have the same baseline as the example title.

ET is the code for "example title." It is to be set with ragged right margins and flush left in the text area (which is to the right of the gutter and 30 picas wide) using the same font as that used for the word EXAMPLE, with a 12-point baselineskip if more than one line is required. There is to be a 33-point baselineskip from the last line before the example title to the first line of the example title, and an 18-point baselineskip from the last line of the example title to the first line of the example text (which is set in the basal font, cmr10). Furthermore, there is to be a 24-point baselineskip from the last line of the example text to the first line of the solution text. However, if there is no solution, then there is to be a 6-point solid color square, base aligned with the last line of the example text and flush right in the text area, followed by a 1-point-high by 30-pica-wide color rule set 9 points, baseline to baseline, below. There

is to be a minimum 30-point baselineskip below this color rule to the basal text.

*Note:* The designer also provides *layouts*, which present a graphical display of the written specs.

Once we and the publisher perused the specs and layouts, potential modifications were discussed and referred to the designer. Then the designer drew up a revised set of specs, taking into account all agreed-upon changes.

## Implementation of the Design Specifications: Writing the Macros and Obtaining the Sample Pages

Upon receipt of the revised specs, we began writing the macros. We aimed to include all the macros that would be needed for the entire book. In retrospect, however, a more realistic goal would have been for the initial set of macros to cover all of the design elements, and to write additional macros on the fly.

After completion of the (original) macros, we typeset a document that would show how the various design factors actually looked on paper. This document was output on a 300-dpi laser printer and was called the *laser sample pages*. The laser sample pages were sent to the publisher for inspection. As might be expected, we and the publisher found several design items that sounded good in theory but did not work out well in practice. So, it was back to the designer for a final revision of the specs.

When we received the final version of the specs, we first effected the necessary changes to the macros. Then we re-TExed the sample-page document and printed the revised copy on our laser printer. At this point, we also prepared a floppy disk containing the latest dvi file for the sample pages. The floppy disk and revised laser sample pages were returned to the publisher.

The publisher then arranged for the creation of the *printed sample pages* to give a true picture of what the book would look like. To obtain the printed sample pages, the same steps were taken on a small scale that would be taken on a much larger scale for the final book:

- The dvi file was processed using a phototypesetter to produce the repro (high-resolution output on specially coated paper).
- Repro was shot and separated for color breaks.
- Art, supplied in film form, was integrated.
- Plates were made.

- Text was printed on the designated paper (in this case, 45# New Era Matte).

It was a thrill for us to see the printed sample pages—we could now imagine the appearance of the final book, although it was over a year away. There were some minor changes that occurred to us and the publisher after reviewing the printed sample pages. The required modifications were made to the macros and we were ready to begin the typesetting of the book.

## Typesetting the Manuscript

While typesetting the manuscript, we kept a printout of the source code for the sample pages handy for reference purposes. This made it easier to remember which control sequences did what. Of course, we also had a printout of the macros available to consult whenever necessary.

We found that for the initial typesetting (which produced the galley proofs) it was convenient to include exactly one section per file. Thus, we named the files by chapter number and section number (e.g., `2-3.tex`). At the end of each file, we employed a macro called `\enddocument` which printed the values of the various registers used to keep the chapter number, section number, chapter title, example number, etc. Those values were then input at the beginning of the next file.

The design called for the exercises to be double columned in nine-point type. But we followed the traditional method of not multi-columning at the galley stage. This saved time and allowed more space for marking corrections. It is important to note, however, that we did set the exercises in nine-point type using the 18-pica width specified for each column of the double-columned text.

As we typeset the manuscript, we often found it necessary to write new macros, especially macros for complex formulas and displays that occurred numerous times in the text. These macros were added to the file `is3macs.tex`, the macro file for the book, as they were written.

Since the majority of the art used for the book was “pick-up” from the second edition, we made no attempt to do the art electronically. The design called for each piece of art to be displayed between two half-point horizontal rules (either 30 or 38 picas wide, depending on the width of the art) with nine points of space beneath the top rule and above the bottom rule. So, in the macro for the figure legend and these rules, we allowed for a parameter specifying the height of a figure. Then

the required space was allocated automatically. In a few instances, some changes were made to the height of figures. These changes were incorporated into the source code before dummying.

## The Galley Proofs (Laser Output)

The first hard copy of the typeset manuscript was done on an Apple LaserWriter Plus (300-dpi). Carol and I both proofread this copy, marked corrections and changes, modified the source code appropriately, and printed out a revised copy. This was done on a section-by-section basis to minimize any fatigue that might ensue from continual typesetting or proofreading.

We sent final galley proofs to the sponsoring editor in batches of three or four chapters. He then made photocopies of the proofs and sent them to the reviewers. It was a tremendous advantage to have the reviewers see the text in a form that showed the design of the book, for it provided them an opportunity to comment on the design before the book was printed. In the traditional method of production, the reviewers usually see only a typed version of the manuscript. Any problems with the design not discovered by the author or publisher are there for the duration of the edition!

Although it is propitious to have the reviewers examine the text in a form displaying the design, one might consider it dangerous to do all of this typesetting prior to the reviewing process; however, in this case, it wasn’t—for several reasons. First, the book was in its third edition and so considerable reviewing had already been done in previous editions. Second, the publisher had arranged for extensive pre-revision reviewing with the idea that most of the major issues would be resolved before the galley proofs were set. And, third, because we were using `TEX` and the original typeset text was not dummed, it really wasn’t that much of a problem to make even extensive revisions.

## Final Text Review, Revision, and Dummying

When all the reviewers had returned a given batch of chapters to the sponsoring editor, he forwarded a copy of their comments and suggestions to me. The editor and I discussed the reviews in detail and decided on final revisions. Subsequently, I went to work making the necessary changes and Carol altered the source code as required.

After all of the final revisions had been completed, we commenced *dummying*. This is the stage

in which the final pages are formed. There were many details to attend to during the dummying stage—so many that we decided to make up check lists to ensure that we didn't forget anything.

Actually, before we began dummying, we measured the final art (or art dummy, in some cases) to make absolutely sure that all was as it should be. We also took care of some last-minute modifications and checked that the miscellaneous corrections marked on the galleys had been executed.

As mentioned earlier, we did not double column the exercises during the galley stage but, of course, we needed to do so at this stage. To make the transition easy, we defined the following two pairs of macros, `\beginsc` and `\endsc`, to begin and end single-column exercises, and `\begindc` and `\enddc`, to begin and end double-column exercises. The two pairs of macros were identical in every respect, except that the first pair typeset the exercises in single-column format (18 picas wide), and the second pair typeset the exercises in double-column format (two 18-pica columns, with a 2-pica gutter). When we were ready to dummy, we simply changed from the first pair of macros to the second pair.

We used a variation of `\midinsert` to handle the placement of tables, figures, and computer printouts. Although the placement is done automatically, changes in text must be made to account for referencing whenever an insertion does not fall in its natural position.

There were other items that required consideration during dummying. For example, the design specified that our *procedure boxes* be color screened. This called for special treatment when a procedure split from one page to the next.

## The Page Proofs (Phototypesetter Output)

Once a chapter was dummed, we copied the dvi files onto a floppy disk. That floppy disk was sent to the publisher along with hard copy (done on our laser printer). The publisher, in turn, made copies of the hard copy and sent the floppy disk and a copy of the laser output to the company that was doing the phototypesetting.

After the publisher received the repro, photocopies were made which, for convenience, we will call the *page proofs*. Page proofs were sent to us and to two proofreaders. The proofreaders also received a copy of the laser output just in case they couldn't read something on the page proofs or there appeared to be some problem on the page proofs. It should be emphasized that the proofreaders' job

was to peruse the page proofs, not the laser output. This was because we wanted the proofreaders to check what would eventually constitute the pages in the book.

Theoretically, the repro should be identical to the laser output except for the difference in resolution. However, that doesn't always happen in practice, so care must be taken. In our case, we found the first ten chapters of the repro to be an exact replication of the laser output; but, in the repro for Chapter 11, we found some strange things indeed: All of a sudden, vertical rules were missing, kerning was often incorrect, and footnotes extended into the margin area. This caused everyone great concern. Fortunately, however, the problem turned out to be simply that the dvi files had been processed using a different computer than previously and there were some compatibility problems. We went back to the other computer and everything worked out fine.

We didn't expect the proofreaders to find too many errors since the text had already been scrutinized. But we had two excellent proofreaders and they did find items that required correction. Those corrections were made by Carol, who then sent in new dvi files as required. She also constructed a chart showing which pages in each dvi file needed to be rerun.

## The Blues Stage and Beyond

When the final repro for a chapter was ready, it was sent, along with the corresponding art, to a pre-press house. That house had responsibility for shooting the repro and separating for color breaks; integrating the art, which had been supplied to them in film form; and stripping the film to the printer's imposition.

We and the publisher each received a set of *page blues*. The page blues are proofs of the negatives that show what the final pages will look like. In making the blues, the black portion of the page is overexposed and thereby shows up in a darker shade of blue than the color portion. This allows for verification of the color separation.

On the blues, we checked the color separation, looked for any stray marks that required cleaning, and verified the figure placement (traditionally, this last item is done in page proofs). Once all corrections marked on the blues had been done, the imposed film was sent to the printer. Plates were then made and the book was printed.

## Comparison of $\text{\TeX}$ with the Traditional Method

Since the introductory statistics book had been produced twice before using the traditional method, I would like to compare, from this author's point of view, the traditional method with the  $\text{\TeX}$  method. To begin, I should say that, personally, I truly enjoy the production phase of a book — from the design stage through to the blues stage. Furthermore, I like writing the macros for the design. Thus, my comments will undoubtedly be somewhat biased.

Probably one of the most compelling reasons for an author to use  $\text{\TeX}$  is that by doing so he or she maintains almost complete control over the production of the book until page proofs (phototypesetter output). Prior to the page proofs, the author is essentially free to make whatever changes that are desired. The publisher really doesn't care whether the author makes changes here and there as long as they enhance the text and are not an expense borne by the publisher.

Another good reason for an author to use  $\text{\TeX}$  has to do with proofreading. Using the traditional method, many authors (me included) must proof the text three times: once each for the manuscript, galley proofs, and page proofs. On the other hand, with  $\text{\TeX}$ , it is probably only necessary to proof the text *at most* twice; and once might suffice if the original version does not require extensive revision, as was the case with the introductory statistics book.

Whether it takes more time and energy on the author's part to produce a book using  $\text{\TeX}$  really depends on several factors. For example, with an excellent compositor, it may take both less time and less energy with the traditional method; but a poor compositor can significantly increase the time and energy that an author must expend (not to mention the added frustration).

A possible advantage of the traditional method over  $\text{\TeX}$  might arise when considering the intensity of the project. I am referring to the fact that when an author uses  $\text{\TeX}$ , there are rarely any breaks in the action. With the traditional method, however, the author generally gets a respite between the completion of the manuscript and its copy editing, between submission of the copy-edited manuscript to the compositor and receipt of the galley proofs, and between the galley proofs and the page proofs.

All in all, for me the choice between  $\text{\TeX}$  and the traditional method is clear—I choose  $\text{\TeX}$ . But, of course, each author will have to balance the pros and cons of using  $\text{\TeX}$  based on his or her own personal experience.

## Bibliography

- Adams, Robert. "Problems on the  $\text{\TeX}/\text{PostScript}/\text{Graphics}$  Interface." *TUGboat* 11(3), pages 403–408, 1990.
- Knuth, Donald E. *The  $\text{\TeX}book$*  Reading, Mass.: Addison-Wesley, 1984.
- Weiss, Neil A., and Matthew J. Hassett. *Introductory Statistics, third edition*. Reading, Mass.: Addison-Wesley, 1991.

# Authors New To $\text{\TeX}$ Publish a TextBook With a Publisher New to $\text{\TeX}$

Samuel E. Rhoads

Information and Computer Science, Honolulu Community College, 874 Dillingham Boulevard, Honolulu, HI 97817  
(808) 845-9277  
Internet: [sam@uhccux.uhcc.hawaii.edu](mailto:sam@uhccux.uhcc.hawaii.edu)

## Abstract

This paper describes our adventure of writing a textbook using  $\text{\TeX}$  and  $\text{\LaTeX}$ , and in working with a publisher, William C. Brown, Inc., who had not worked successfully with  $\text{\TeX}$  in the past. The paper discusses the learning process we went through in learning  $\text{\TeX}$ , in working with a publisher new to  $\text{\TeX}$ , and in writing a textbook. It is hoped that by sharing our experiences, other authors and publishers will realize how easy producing a high-quality book can be, and perhaps some of the mistakes we made can be avoided.

## Introduction

In April of 1987, I received a phone call from Mike Gearen who teaches computer science at Punahoa School in Honolulu. He had seen a teacher's guide for the Advanced Placement Computer Science course that I'd written, and he wanted to know what programming-in-Pascal textbook I used in my classes. He had been unable to find a book he liked and thought maybe I'd found one I liked. We met a few days later and discussed writing our own book. I'd heard of  $\text{\TeX}$  and thought that using  $\text{\TeX}$  would make the writing of a book easier. Thus began our experiences with  $\text{\TeX}$  and with publishing a textbook.

It's an understatement to say that the four years since that meeting have been exciting. We have seen many frustrating times, but the rewarding and exciting times far outweigh the frustrating ones. I still get excited everytime I see a beautiful page of print coming out of my laser printer.

We purchased our first copies of  $\text{\TeX}$  and  $\text{\LaTeX}$  from Addison-Wesley in the summer of 1987, and started writing. The start-up time in learning  $\text{\TeX}$  and  $\text{\LaTeX}$  was much shorter than I'd feared. In what now seems like no time at all, we were preparing pages of the book. In fact, we had enough done by the end of the year that we used our laser printer output as a textbook in our classes in the spring semester of 1988. (Although we used  $\text{\LaTeX}$  for the preparation of the book, we always have thought of it as using  $\text{\TeX}$ . Unless the distinction is important, for the rest of this article, I'll say " $\text{\TeX}$ " instead of saying " $\text{\TeX}$  and  $\text{\LaTeX}$ .)

We sent a prospectus and a couple of chapters (done in  $\text{\TeX}$ , of course) to several publishers in late '87 and waited to hear from them. Two of the publishers liked our prospectus, and offered us contracts. In both cases we made it very clear that we wanted to use  $\text{\TeX}$  to prepare the manuscript. Neither publisher had had a successful experience with  $\text{\TeX}$  before. One publisher had had no experience at all with  $\text{\TeX}$ , and the other had only had one, unsuccessful attempt. (More on the unsuccessful experience later.) Both publishers initially discussed what they called "electronic submission of manuscripts." To them, this meant sending the manuscript to them on a disk so it would not have to be typed in again. By this time we were so impressed with the appearance of the book that we were brave enough to insist that the book be done in  $\text{\TeX}$  as a condition of signing the contract. After thinking about the two publishers, and trying to decide which offer was better, we decided to go with William C. Brown Publishers, Inc., and signed the contract on February 3, 1988.

By this time a rough version of the majority of the book had already been completed, and, as mentioned above, we were using it as a textbook in our first course in programming during the spring semester, 1988. We were still working on the later chapters and developing our  $\text{\TeX}$  skills. Rough versions of these chapters were completed in time to hand out to our classes by the time they were needed.

## TeX and L<sup>A</sup>T<sub>E</sub>X Skills

As our work on the manuscript progressed, we decided it would be convenient to define some macros. Our first macro was designed to simplify the printing of *<block>*. We saw that we could accomplish this by typing: `\$\\langle\$\\{\\it block}\\$\\rangle$` everytime we needed it, but soon realized that there was an easier way. Looking back, it seems like a minor victory but we were pleased when we learned that we could define the L<sup>A</sup>T<sub>E</sub>X macro:

```
\newcommand{\block}
{\\langle\\{\\it block}\\$\\rangle$}
```

and from then on whenever we wanted to see *<block>* in our book, we only needed to type `\block`.

As would be guessed, this quickly led to many other macros. We busied ourselves defining macros that were intended to make our typing easier. We decided to collect these macros in a file, `macros.tex`, and to input the file at the beginning of each `.tex` file. It was easy to get carried away with this. In looking at `macros.tex`, I see a macro: `\newcommand{\real}{\\tt real}`. Even though we had defined this macro, we seldom used it; we just typed `\\tt real` rather than `\real`.

**Fonts.** In our book we spend a great deal of time discussing algorithms. We stress the development of an algorithm prior to the writing of the code. We decided that we should use a special font to represent an algorithm, and we started looking through *Computer Modern Typefaces* for a font to use. We wanted a typeface that reminded the reader of handwriting — algorithms probably should be handwritten rather than typed — and we found `cmff10` and `cmfi10`. Knuth [*Computer Modern Typefaces*, page 28] calls these fonts Computer Modern Funny Font and Computer Modern Funny Italic, respectively, and says that `cmfi10` is not quite as “hilarious” as `cmff10`. I’m not sure whether Knuth expected people to actually use these two fonts, or was just defining them for fun. In any case, we don’t find `cmfi10` funny at all; we like it. We tried `cmff10`, but it proved too hard to read. I wonder if any other use for either `cmfi10` or `cmff10` has been found by other authors.

The definition for this font — we call it `\al` for “algorithmic” — was placed in `macros.tex`. Thenceforth, to produce: *Store true in Found*, we just typed: `\al store True in Found`.

While this worked fine for short examples of a step or two, when we wanted a complete algorithm to appear, we encountered a harder problem: We wanted an algorithm to fit completely on a page

and not to be broken where L<sup>A</sup>T<sub>E</sub>X decided to break the page. We wanted the algorithm to be printed in the algorithmic font we’d chosen. We also wanted to be able to indent the examples of algorithms and code that we presented in the book, and we wanted the indentation to be uniform. This was one of the hardest L<sup>A</sup>T<sub>E</sub>X problems we had to solve. As with many problem solutions, I’m not sure our solution was the best solution — in fact, I’m sure it isn’t — but it works and we’ve used it since. We played with different ideas for quite awhile, and finally developed the following two L<sup>A</sup>T<sub>E</sub>X definitions:

```
\newcommand{\balg}
{
\par
\al
\begin{minipage}{\textwidth}
\begin{tabbing}
123\=123\=123\=123\=123\=123\=\kill
\mbox{\ }\\
\end{tabbing}
\end{minipage}
\rm
\par
}

\newcommand{\ealg}
{\mbox{\ }\\
\end{tabbing}
\end{minipage}
\rm
\par
}
```

Then whenever we wanted to write a complete algorithm, we just coded:

```
\balg
START OF Algorithm
.
.
.
END OF Algorithm
\ealg
```

to produce:

```
START OF Algorithm
.
.
.
END OF Algorithm
```

The `minipage` environment forces L<sup>A</sup>T<sub>E</sub>X to keep the entire algorithm on the same page; this sometimes caused ugly page breaks that we had to fix by hand. The `tabbing` environment allowed us to indent our algorithms by just typing `\+` whenever we wanted to move to the right one level of indentation, and `\-` when we wanted to move

to the left one level. The `\hbox{\ }` produces an empty line. The `\al` changes to the algorithmic font, and the `\rm` changes back to roman.

The appendix contains an example of how we typed an algorithm for a function that returns the greatest common divisor of two positive integers, and the algorithm as it appeared in our book.

**Figures and Pictures.** We made great use of L<sup>A</sup>T<sub>E</sub>X's figure and picture capability. We defined figures and pictures to be things that floated. Figures and pictures always had captions; indeed every picture was in a figure. As an example of how these figure/pictures were coded, I've included a very simple one in the appendix.

Both of us have decent math backgrounds so the geometry and algebra that were required to get the coordinates of the rectangles, lines, vectors, ovals and circles correct wasn't hard. I imagine that for less mathematically trained writers this would prove intimidating, and thus I would expect other authors to turn to more powerful aids. Indeed, as easy as we found it, it would be nice to know better ways to get the pictures of syntax diagrams and trees drawn.

One figure in particular needs to be discussed. Early in the book we give a skeleton of a complete Pascal program. It was supposed to look like this:

```
program <programname>(<filelist>);
<block>
```

**Figure 4.1** The Form of a Pascal Program

Instead, it looked like this:

```
program <programname>(<filelist>);
<block>
```

**Figure 4.1** The Form of a Pascal Program

We had decided to put all examples of algorithms and all figures in a different color (blue). We had also agreed to have the final pages of the book done on a high-resolution printer by ArborText, Inc., in Ann Arbor, Michigan. In order to do the second color, the publisher had to "cut out" some of the final copy and print it in blue. (I'm not sure how this is done, so I'm being deliberately vague.) When Wm. C. Brown printed it, they must have thought

the period was a flyspeck, or something, and they left it off. That wouldn't have been so bad except on the next page of the book, we wrote, "Did you see the period after the *(block)* in figure 4.1? It's easy to miss, but it's necessary." It sure was easy to miss, it wasn't there.

Wm. C. Brown must be given credit. When they learned of the missing period, they had someone go through the warehouse and put a little dot in every one of several thousand books.

## Other Design Issues

**Illustrations.** We have always been taken with Duane Bibby's illustrations in *The T<sub>E</sub>Xbook* and decided to find an artist to draw illustrations of a computer programmer, a user, and a personified computer for our book. (We considered trying to find Duane Bibby himself, but couldn't muster up the nerve.) After considerable searching, we finally contacted a local caricaturist, Katie Ralston, and had her draw the illustrations. (Wm. C. Brown calls these illustrations "cartoons.") I mention the illustrations since they were the only things in the book not done by T<sub>E</sub>X. The publisher had to insert them into the final pages prepared by ArborText. We did, of course, leave room for them in the proper places.

**Style Files.** The design staff at Wm. C. Brown wanted a few changes made in the design of the book. In order to implement these changes, we had to modify the style files. We didn't want to change the `.sty` files themselves, so we made copies of the `book.sty`, `bk10.sty`, `bk11.sty`, and `bk12.sty` files, giving them different names, and made the modifications to those files. Rather than describe all these changes, I'll just describe one, as an example.

We used the `book` style with 10-point type. In this mode, L<sup>A</sup>T<sub>E</sub>X causes the running heads to be printed in ten-point, uppercase italics. The design people wanted nine-point upper- and lowercase italics. We searched through our renamed style files until we found the following definitions:

```
\def @evenhead{\rm \thepage
\hfil \sl \leftmark}
\def @oddhead{\hbox{} \sl
\rightmark \hfil \rm \thepage}
```

We modified these two definitions to:

```
\def @evenhead{\rm \thepage
\hfil \small \sl \leftmark}
\def @oddhead {\hbox{} \small \sl
\rightmark \hfil \normalsize \rm \thepage}
```

The `\small` that we added simply changed the running head to nine point. The `\normalsize`

changed it back to ten point before printing the page number on odd-numbered pages.

That took care of changing the size of the letters. Now we had to see if we could have it print the head in upper- and lowercase rather than all uppercase. Again searching through our renamed copy of *book.sty*, we found this definition:

```
\def\chaptermark##1{\markboth
  {\uppercase
  {\ifnum\c@secnumdepth>=0\@chapapp\
  \thechapter. \ \fi ##1}}{}}
```

We finally figured out that the `\uppercase` was turning the name of the chapter into all uppercase letter. (Actually, it was easy figuring out what it was doing; the hard job was finding it.) By changing the definition to:

```
\def\chaptermark##1{\markboth
  {{\ifnum\c@secnumdepth>=0\@chapapp\
  \thechapter. \ \fi ##1}}{}}
```

that is, by just removing the `\uppercase`, the running head was in upper and lower case.

We made several other changes to the style files. We changed the margins so that odd and even pages would print with margins that would make it easy to cut the book down to a  $9 \times 7\frac{1}{2}$  format. We changed the definition that printed the caption in figures to print the word **Figure** in bold face. All of these changes took time to figure out what to do, but none were particularly hard to figure out.

## Comments From the Publisher

I asked the woman who copyedited our book to send me some feedback concerning our use of *T<sub>E</sub>X* to print the book, explaining that I was writing this article. She forwarded my request to Wm. C. Brown's electronic text coordinator, the person responsible for working with authors "preparing manuscripts on disk." Here are her comments regarding Wm. C. Brown's experiences with *T<sub>E</sub>X*:

WCB's initial experience with *T<sub>E</sub>X* was not a successful one. About three years ago, Kendall-Hunt had an author working on *T<sub>E</sub>X* who produced a math book. Manufacturing was persuaded to purchase the *T<sub>E</sub>X* program, as it seemed to be becoming the software of choice for those who wished to produce texts containing math and other types of equations. After several unsuccessful attempts in-house to tailor the files to be compatible with our typesetting system, manufacturing resorted to going to an outside preparer (an engineer)

who produced camera-ready pages from laser output.

Approximately a year later, WCB received a computer programming book that was done in *L<sub>A</sub>T<sub>E</sub>X*. Outside suppliers using *T<sub>E</sub>X* (and its various versions) had been busy writing software to make the *T<sub>E</sub>X* program compatible with traditional typesetting systems. We were able to find an outside vendor who produced high-resolution, paged output from a typesetter.

The authors worked with in-house staff on questions of design, layout, and typography. They were cooperative in making adjustments wherever possible to achieve a pleasing format for the text.

The disadvantages of this were that control of the project went out of house. Also, at that time, the choices of typefaces were limited.

The advantages, however, outweighed these disadvantages. Namely:

1. The authors were willing to make whatever changes were needed because of copyediting, so WCB had no involvement in time or personnel in the updating process. The responsibility for getting "perfect" disks to the vendor was the authors'.
2. The vendor worked directly with the authors and was able to solve any start-up problems.
3. There was no keystroking required in-house.
4. All formatting integrity was maintained.
5. WCB received paged output in approximately three weeks—much less time than "traditional" output.
6. All graphics were in place on the page except for photos and acquired cartoons, minimizing hand keylining time.
7. Corrections were minimal.
8. The cost of the project was much lower than if it had been handled traditionally.

The project was such a success that we will continue to consider sending any *T<sub>E</sub>X* project to an outside vendor that is capable of sending the files to high-resolution output. There are at least two projects that will be handled that way this year. Progress continues to be made in expanding the capabilities in regards to typefaces and output possibilities.

*T<sub>E</sub>X* is also offering various forms of its own product to attract and expand its

market potential. We continue to monitor these products as there is interest in our math area about *T<sub>E</sub>X* products. Many of our math authors and math ancillary authors are using *T<sub>E</sub>X*. We may want to test the possibility of having manufacturing using the software in typesetting.

The one drawback may be that *T<sub>E</sub>X* does not give the same high quality that may be expected in upper-level textbooks, and it is difficult to integrate it with traditional typesetting modes. Also, one must be careful regarding vendor claims to be able to print pages for nominal charges. There are hidden traps in these offers.

**Response.** While I don't intend to discuss each of these comments, there are a couple of things I want to say: The "computer programming book" she refers to is, of course, our book. The "vendor" who produced the high-resolution paged output was ArborText, Inc.

It's clear that doing a book this way causes things to be done differently than they have been done in the past. In our case, the copy editor made changes on the hard copy we sent her, and then sent these changes back to us to make. In a few instances, we didn't want to make the change. I suspect that this might have caused some problems.

She has the impression that *T<sub>E</sub>X* is only good for math and "math ancillary" textbooks. I hope, and suspect, that the future of *T<sub>E</sub>X* will prove it to be the best choice for books of all types.

She indicates that the choices of typefaces were limited. That may have been true, and it may still be true; I'm not sure. In any case, I would have chosen Computer Modern if I had a choice, so I'm glad that there weren't others available.

I don't know what she means by "All formatting integrity was maintained."

It seems clear that *T<sub>E</sub>X* made the project both faster and cheaper for Wm. C. Brown. The final sentence is puzzling, though. I don't know whether she is referring to the first experience or the second, and I don't know of any hidden traps. The process of sending the disks to ArborText and having them send high-resolution hard copy to Wm. C. Brown seemed to go very smoothly.

I am concerned about the "drawback" regarding quality. Here I must take the blame. I'm convinced that *T<sub>E</sub>X* is capable of producing textbooks of the highest quality. If there was a limitation perceived by Wm. C. Brown, the limitation was in my ability, not in *T<sub>E</sub>X*'s. I will have to try to make that clear to them.

## Conclusion

The experience of writing a textbook was quite a challenge. It was the first major book either of us had written, and we didn't know much about the process. By doing it in *T<sub>E</sub>X*, I suspect there are many things we never had to learn.

I have a friend who wrote another textbook at the same time we were writing ours. His book was published by a different publisher, and his publisher re-typed the entire book—even though he had sent them disks. He complains to this day about typographical errors that he finds in the book that were not there in the version he sent them. He had a chance to proofread his book, but he was not able to find them all. Our experience was quite different from his; once we finished a page, we knew what the final version would look like, and typographical errors could not creep in through the typesetting process. I wouldn't think of trading places with him.

A good part of the fun was in learning *T<sub>E</sub>X*; another was in reading *The T<sub>E</sub>Xbook*. The interchange between us and the various editors at Wm. C. Brown was a learning experience that we won't soon forget.

After all is said and done, and even though there were times that we got frustrated, it's obvious to us that *T<sub>E</sub>X* is the way to go. Had we not had *T<sub>E</sub>X* to use to prepare the prospectus and the preliminary versions that our students used, I doubt that it would ever have gotten finished. Indeed, without the beautiful prospectus, it might never have been accepted for publication.

I'm convinced that any author can learn enough *T<sub>E</sub>X* and/or *L<sub>A</sub>T<sub>E</sub>X* to write their articles, books and papers; it's not necessary to become a *T<sub>E</sub>Xpert* to develop beautiful results. A fast computer—I'm now using a Sun workstation—and a previewer make life easier. I cannot imagine writing without *T<sub>E</sub>X*.

## Bibliography

- Knuth, Donald E. *The T<sub>E</sub>Xbook*. Reading, Mass.: Addison-Wesley, 1986
- Knuth, Donald E. *Computer Modern Typefaces*. Reading, Mass.: Addison-Wesley, 1986.
- Rhoads, Samuel E., and Michael V. Gearen. *Disciplined Programming Using Pascal*. Dubuque, Iowa: William C. Brown Publishers, 1990.

## Appendix

This is how we typed the algorithm for GCD:

```
\balg
START OF GCD$(First, Second)$\\
Precondition: First and Second are positive integers\\
\mbox{\ }\\
store the remainder of First $\div$ Second in Remainder\\
loop while Remainder $> 0$+\\
    store Second in First\\
    store Remainder in Second\\
    store the remainder of First $\div$ Second in Remainder$-$\\
return the value stored in Second\\
\mbox{\ }\\
END OF GCD
\ealg
```

And this is what the algorithm looked like in the textbook:

```
START OF GCD(First, Second)
Precondition: First and Second are positive integers

store the remainder of First  $\div$  Second in Remainder
loop while Remainder  $> 0$ 
    store Second in First
    store Remainder in Second
    store the remainder of First  $\div$  Second in Remainder
return the value stored in Second

END OF GCD
```

The following is an example of a simple figure/picture done in L<sup>A</sup>T<sub>E</sub>X. It draws a syntax diagram of the Pascal `while` statement. This example is included just to show, in general, how we constructed such diagrams. Most of the figure/pictures were much more complicated.

```
% whstate.tex  the while statement

\begin{figure}[htb]
\begin{picture}(352,64)
\put( 0, 48){\it while statement:}
\put( 8, 24){\vector(1,0){24}}
\put( 56, 24){\oval(48,16)}
\put( 56, 24){\makebox(0,0){\tt while}}
\put( 80, 24){\vector(1,0){24}}
\put(104, 16){\framebox(88,16){\it Boolean expression}}
\put(192, 24){\vector(1,0){24}}
\put(232, 24){\oval(32,16)}
\put(232, 24){\makebox(0,0){\tt do}}
\put(248, 24){\vector(1,0){24}}
\put(272, 16){\framebox(48,16){\it statement}}
\put(320, 24){\vector(1,0){32}}
\end{picture}
\caption{\label{whstate}}
\end{figure}
```

# The “Five Cs”: A Guide to Successful Publication Using T<sub>E</sub>X

Colleen Brosnan

College Book Editorial-Production, Prentice Hall, Englewood Cliffs, New Jersey 97632  
(201) 592-2312

## Abstract

From the perspective of working at Prentice Hall College Division, my paper will cover the importance of the five C's: early Contact with your publisher, Consistency of macros, Compromises on issues such as design, Constraints of time and cost, and Communication, which is probably the most important.

I am manager of Technical Manuscripts in the College Book Editorial/Production Department at Prentice Hall and I would like to take this opportunity to share with you the experiences that we at Prentice Hall have had with authors who have submitted manuscripts prepared in T<sub>E</sub>X.

Until a few months ago, I was Production Manager of the Computer Science and Engineering team. My team handled the majority of T<sub>E</sub>X manuscripts that were published by the College Division, so I think I've seen it all—good and bad.

The Prentice Hall College Division has been working with authors who have been preparing their manuscripts in T<sub>E</sub>X for about five years and has published about 100 titles.

Over the years, we have encountered authors at all levels of expertise in T<sub>E</sub>X. Some have been extremely proficient in the use of T<sub>E</sub>X but have not been able to create acceptable page layout. For those, we send the author's files to a compositor who formats the files according to our page makeup specifications and inserts copyediting changes. Other authors are able to do it all—providing us with camera ready copy, even with separations for two-color books. And others provide us with files that are virtually unusable for a variety of reasons—we send those manuscripts to a compositor to be keyboarded from scratch.

We have learned a great deal during those years by working closely with our authors and compositors. The result of this collaboration is that we have developed what I like to call our “Five C's”—the keys to successful, painless (for both author and publisher) publication of T<sub>E</sub>X documents.

These “Five C's” are early *contact* with your publisher, *consistency* of your macros, *compromises* on issues such as design, *constraints* of time and cost, and, of utmost importance, *communication*.

## Contact: Better Now Than Later

In an attempt to avoid unnecessary work, you should always contact your acquisitions editor before beginning any formatting of your manuscript. Your idea of a great design for your book may not be the publisher's—especially if the trim size you've selected is not appropriate for the market. The hours you've spent in creating your design will be wasted if your design won't be used. At Prentice Hall, we spend a great deal of time analyzing the marketplace and select designs that will be cost effective (use of tints, for example, may be quite costly to produce), geared to the audience (an introductory computer text may warrant liberal use of highlighted text or boxed material that would be distracting in a graduate-level programming book), and conform to the style of book that professors and department heads have told us works best for their courses.

If you feel you want to take a stab at preparing camera ready copy, ask your acquisitions editor to send you a set of book specifications for you to follow. Possibly the typesetting language that we use in writing our specifications will be as much Greek to you as your macros are to us. If so, call your production person and ask for a translation.

After you've formatted a sample chapter that contains the most representative elements in your book, send it to us in both hard copy and electronic form. If you're not sure whether we want your files on a cartridge, tape or floppy disk, call us and we'll advise you. Some compositors work only from floppy disks—others can use all types of media.

Your sample chapter will be reviewed by a production manager and art director for style and quality. Based on their recommendations, your acquisition editor will decide whether your formatting meets our standards for publication in that particular market.

At times, even though the formatting is close to acceptable, we feel that more "fine-tuning" is needed. In those cases, we send the chapter back to you with instructions for improving the design or page layout. From the feedback we receive, we can get a sense of your expertise in this area. By the same token, you can decide, based on our comments and suggestions, whether the amount of work that will be expected is a task that you want to undertake.

If we decide that you will provide camera copy, do not format your entire book yet. Send us a double-spaced manuscript that we can use for copyediting. After you enter the copyedits, then you should do your final page makeup.

This initial contact is only the beginning of our effort at P-H to ensure that your *T<sub>E</sub>X* manuscript will be converted to a bound book with the least amount of effort in the shortest time possible.

If we—or you—decide that page makeup is best left to the "experts", we will send your electronic files to one of our compositors who specialize in *T<sub>E</sub>X* for a test to determine if the files can be used with a minimum of time and effort.

Our compositor will go through your macros and set selected elements, according to the specifications we have provided. You should give us guidelines on which elements or special characters you want to see set. These sample pages will help the compositor to identify any "glitches" in your macros that you will be asked to correct before submitting the entire manuscript. These sample typeset pages will be sent to you to review to make sure that elements have been typemarked correctly—for example, whether computer code has been set in nonproportional fonts and whether headings have been given the proper order of importance. You should proofread these samples carefully to verify that any special characters you have used have been translated properly.

## Consistency

Consistent preparation of macros is one of the most critical issues determining whether your electronic files can be used by our compositors. "Keep It Simple" is the advice that all of our compositors give. Don't be so concerned about writing macros that "look good" at the expense of macros that "work well."

Even the most ambitious among you may find that keyboarding your entire manuscript in *T<sub>E</sub>X* is simply too time-consuming. If this turns out to be the case for you and you have to turn the job over to your assistant or several graduate students—or

if you have co-authors who are involved in your project—be sure to give them clear instructions on what macros you have used and what hard coding you have done so they can duplicate your work.

Whenever possible, use the default macros available in the version of *T<sub>E</sub>X* you choose. If you need to make some modifications—perhaps a special macro or two, or some time-saving string definitions—put these into a file of their own, using the \input command to read it in during processing. Be sure to put any customization files onto your tape or floppy so they are available. The file should always include a "read me" file explaining the macros and identifying any hard coding that has been added. Compositors spend an inordinate amount of programmer's time trying to unravel several different sets of macros for the same elements. Time spent by compositors trying to figure out what an author has done is a poor utilization of their resources.

A "clean" file should contain macros that can easily be converted to the macros that the compositor uses to implement the publisher's design. So, if there are elements of your manuscript that occur frequently, such as theorems, examples, or quotations, develop macros for them instead of putting space around them or putting them into other fonts. For example, you could use a simple macro that would add space before a theorem, set the heading in bold, print the theorem, and add space after it. If uncomplicated macros such as these are used consistently throughout your manuscript, our compositors will have a much easier job of implementing our design specifications and, in turn, will be able to produce finished pages faster.

Although the default font in *T<sub>E</sub>X* is Computer Modern, at Prentice Hall we prefer to use Times Roman in typesetting our books. Our compositors have redefined most of the standard *T<sub>E</sub>X* font calls to conform to our specifications. Therefore, keep your personal font definitions to a minimum to enable our compositors to translate to our fonts more quickly and easily. However, we do have one exception. We have continued to use Computer Modern for all math because of spacing problems in the conversion. Several of our compositors are working on this problem and we may be using Times Roman for that as well in the near future.

Remember that our compositors base their estimate of the work involved in your project on the files you submit in the beginning. After your sample chapter has been test-run and the compositor has given some feedback on the usability of your electronic manuscript, you will be asked to send in your entire manuscript in both hard copy

and electronic form. Be sure to save each chapter as a separate file; extra long chapters should be broken into two files.

Always send in two sets of the electronic files as we send your manuscript and files to two compositors for bid. We don't routinely duplicate tapes or disks. Failure to send in duplicates will only slow down production of your book as one compositor will have to review and return your files before the other compositor can do the same. We are aware that tapes are expensive, and we will try to return any unused tapes to you. However, the cost of tapes is minor compared to the cost of delaying the in-stock date of your book.

The complete manuscript, which should incorporate any suggestions made by the compositor who looked at your sample chapter, is then sent out for a thorough castoff and estimate by our two compositors. A decision regarding which compositor is awarded the job is made based on the time and amount of work they feel is needed for your project. So whatever you do, don't decide to change your macros—or switch versions of T<sub>E</sub>X, or even to a different word processing package—after you've sent in your original manuscript, unless you notify your production manager. Believe me, this has happened, and has caused an inordinate amount of anguish on the part of our compositor who had spent days trying to figure out why the files couldn't be loaded. Not only did this project cost more than anticipated, it was unnecessarily delayed because the author did not let us know about the switch.

Because macro usage is such an important issue in the decision to typeset from T<sub>E</sub>X files, Prentice Hall has worked with our compositors in developing standard macro packages, complete with documentation. These macro packages are designed to simplify the preparation and production of technical books and will cut time from the final production of your book. Be sure to ask your acquisitions editor about these standard macros before you start on your project to save time for both you and us. These macros will not produce final, single-spaced book pages—rather, they will enable you to print out, on whatever printer you have available, a double-spaced manuscript that we can use for copyediting. After you have reviewed our copyediting, the compositor will then substitute our design macros, implementing our fonts and specifications, to produce the final galley and pages for your book. These compositors also offer technical support if you have questions about the macro packages.

An added benefit to using these packages is that at the end of the production process, if your

electronic files have been used, you will receive not only a professionally produced book but also an electronic file that matches the book for future updates and revisions.

## Compromises

Publishers have certain standards for producing books. Authors have certain standards for their material. Sometimes these two clash. For instance, you may want a lengthy computer statement to be contained on one page. Often this is simply physically not possible so the compositor will break the computer statement at a place required for good page makeup. To avoid inappropriate breaks, send us a list or samples of where computer statements can be broken for the compositor to follow. If you are concerned that the student may not understand that the computer statement continues onto the next page, ask us to insert a "jump" line at the bottom of the page that says "continued on next page." Or you may have equations that are too long to fit within our text column. The choices are to either set those in smaller type (which I don't recommend if you have subscripts that may become unreadable if set smaller) or to break the equation into two or more lines. Our style is to break equations before an operational sign, but you may have different ideas. If you let us know your preferences before we begin typesetting, we can implement them without additional cost or delays. If you see typeset pages and then want to make changes, the cost can be considerable.

You may be used to printing small quantities of your work in a corporate or academic environment where the style, for example, may be to begin a new page for every first-level heading. As book publishers who produce thousands of copies of your book at a time, we have to be concerned about the number of pages your book contains. We try not to have books that have a lot of wasted space—blank pages or pages with only couple of lines of text. Our concern is for the readability of the book—pages that do not follow a logical pattern are difficult for the reader to understand. Book publishing is a competitive business, and books that may be excellent in terms of content may not even be considered as possible adoptions because the format is totally different from what professors are accustomed to seeing. So be aware that the design compromises that we ask of you are not based on arbitrary decisions—rather, there are sound economic and marketing issues involved.

## Constraints

Authors and publishers share a common goal: turning a manuscript into a bound book and in a reader's hands as quickly as possible.

To accomplish this goal, some hard decisions may have to be made.

You may be a whiz at page makeup, can translate our design specifications perfectly, and have access to a high resolution printer. After discussions with your acquisitions and production editors, you have decided to provide your own camera ready copy. Then the project hits a snag. Copyediting changes may be heavier than you anticipated, page layout may become difficult or just plain tedious, or your printer may become unavailable. If any of these scenarios materialize, contact your editors immediately. We can help find reasonable solutions (for instance, we do have service bureaus that can provide high resolution output of your PostScript files). Keeping the project moving may call for a change of plans.

Also, as I discussed earlier, our compositors base their estimate of the cost of producing your book on the amount of work your manuscript needs to turn it into book form.

If you have used macros inconsistently, a programmer may need hours to work through the problems encountered. This is not only costly—a programmer's time can be quite expensive—but time consuming as well. The original schedule we had drawn up for producing your book will have to be scrapped because of these delays. We have asked our compositors to alert us whenever they encounter serious problems with a manuscript before proceeding. In many instances, it is better for the compositor to keyboard your manuscript from scratch. A programmer's hourly rate may be five times that of a keyboarder's. From a strictly economic viewpoint, it's not hard to see which path makes sense. Remember, too, that even if we did decide to have a programmer unravel your macros, a keyboarder is still needed to insert any copyediting changes.

In another scenario, the macros may have been flawlessly prepared and the compositor has no problems converting the macros to our design. But the amount of copyediting is substantial. In those cases, the compositor's keyboarder can actually rekey the entire manuscript more quickly than insert numerous changes to an already existing file.

Often I have heard from authors that the reason they want their files used is that they don't have time to proofread the typeset galley and pages. You always have to proofread galley and pages—even if

your files are used with no intervention. No matter how sophisticated the typesetting system is, the possibility for glitches exists. For instance, a special character could not convert properly or automatic numbering could get turned off inadvertently. We have excellent proofreaders who check your galley and pages, but only you know exactly what should be in your book. Review of galley, pages, and art takes time so let your acquisitions editor and production editor know in advance if you are going to be out of the country or are meeting other deadlines during the times scheduled for review. We will rearrange our schedules whenever possible to work around your other commitments.

Most authors will be unhappy that their *TeX* files are not used, but if they keep in mind our goal—getting the book on the bookshelf quickly and economically—the decision will be easier to understand.

## Communication

In conclusion, I would like to stress the importance of communication. Authors have their desires and expectations. Publishers have their requirements and deadlines. Compositors have their constraints and needs. Unless all three can communicate, there is bound to be some misunderstanding.

From your initial contact with us, don't hesitate to ask questions. We may not know all the answers immediately—for instance, should you send your electronic files on tape? 1600 bpi?—but we will contact the compositor who will be working on your job and find out. If you have a technical question, we may ask our compositor to contact you to resolve it. Our production editors are not *TeX* experts—we leave that to the professionals—but we will guide you to the proper source. Questions about design issues, copyediting preferences, and scheduling problems should be directed to our production editors—that's their area of expertise.

In conclusion, I would like to review again our "Five C's"—*contact, consistency, compromise, constraints, and communication*. By keeping these in mind and working together to solve problems and concerns, authors and publishers will be able to learn from each other, share our knowledge, and smooth the production of your *TeX* document from manuscript to bound book.

Remember, at Prentice Hall, publishing your *TeX* manuscript in the most convenient, cost effective way is our ultimate goal.

# L<sup>A</sup>T<sub>E</sub>X/T<sub>E</sub>X User: A Typist, or Typesetter?

Anita Z. Hoover  
University of Delaware  
002A Smith Hall  
Newark, DE 19716  
Internet: [anita@brahms.udel.edu](mailto:anita@brahms.udel.edu)

## Abstract

The purpose of this paper is to point out that a L<sup>A</sup>T<sub>E</sub>X/T<sub>E</sub>X user who produces documents in camera-ready form is more than just a typist; he/she is a typesetter. These users need to go beyond learning the basics, but not to the point of frustration. There are several issues that should be considered when using L<sup>A</sup>T<sub>E</sub>X/T<sub>E</sub>X for publishing. This paper attempts to look at these issues and share strategies to help those who use L<sup>A</sup>T<sub>E</sub>X/T<sub>E</sub>X for camera-ready publishing.

## Introduction

L<sup>A</sup>T<sub>E</sub>X/T<sub>E</sub>X user: A typist or typesetter? This question has recently become an interesting topic. Is the person who uses L<sup>A</sup>T<sub>E</sub>X and/or T<sub>E</sub>X more than just a typist? I think so! If a person is using L<sup>A</sup>T<sub>E</sub>X/T<sub>E</sub>X to publish a document, following a layout specified by a publisher, this person is more than just a typist. She or he is a typesetter.

From the preface of the *T<sub>E</sub>Xbook*,

...T<sub>E</sub>X, a new typesetting system intended for the creation of beautiful books—and especially for books that contain a lot of mathematics. By preparing a manuscript in T<sub>E</sub>X format, you will be telling a computer exactly how the manuscript is to be transformed into pages whose typographic quality is comparable to that of the world's finest printers....

In the past year, I have spent a large portion of my consulting duties helping graduate students, secretaries and professors at the University of Delaware put together documents that were published using L<sup>A</sup>T<sub>E</sub>X and/or T<sub>E</sub>X. There are two ways these documents can be produced.

### 1. User-Defined Macros

The user must define macros to set up the document to meet a publisher's specifications and then submit a final printed copy. This requires a lot more work for me and a lot more time before the final copy is completed.

### 2. Publisher-Defined Macros

The publisher supplies macros that meet the publication specifications. The user uses these macros to set up the document, and then

either submits a final printed copy or sends in the L<sup>A</sup>T<sub>E</sub>X/T<sub>E</sub>X file. This is a real advantage, as long as the publisher can provide good documentation on how to use the macros. If a lot of time is needed to interpret the use of the macro, then half of the advantage is lost.

## Experience with User-Defined Macros

**Thesis format.** My first challenge was setting up macros in L<sup>A</sup>T<sub>E</sub>X/T<sub>E</sub>X to meet the requirements set by the University of Delaware's Office of Graduate Studies for theses, dissertations, and executive position papers. During this project I realized that many of the specifications had been based on typewritten documents. For example, the document is supposed to be double spaced. It took me three months to convince the Office of Graduate Studies to accept a L<sup>A</sup>T<sub>E</sub>X/T<sub>E</sub>X document that was spaced  $1\frac{1}{2}$  times rather than 2.

Four important points came from this project:

1. Having the macros does not mean that the user does not have to pay attention to the original specifications or guidelines. It is important that the user check the document for correctness. Macros are developed with the intention of being correct, but errors do happen.
2. Users need to be reminded that the macros have been defined to meet certain specifications, and as a result the macros should not be changed. I hear complaints such as, "I don't like the way the document looks." The point is that it does not matter how they think it should look, and altering the macros means the document no longer conforms to the specifications.

3. There needs to be good documentation on how to use the macros. References to which macros fulfill which specifications are important.
4. Examples should be provided whenever possible. Example documents of the input and output are easy ways of showing the organization of the document, how to use the macros, and what they will produce.

**Books.** I have been involved with several publishers that have accepted L<sup>A</sup>T<sub>E</sub>X/T<sub>E</sub>X as the final output for books, but that did not provide macros. Listed below are a few examples:

1. (T<sub>E</sub>X-UNIX)

Karl W. Böer. *Advances in Solar Energy*, 6 volumes, Plenum Press, 1982–1990, ca. 500 pgs.  
 Karl W. Böer. *Survey of Semiconductor Physics*, Van Nostrand Reinhold, 1990, ca. 1400 pgs.

2. (L<sup>A</sup>T<sub>E</sub>X-PC)

Thomas K. Gaisser. *Cosmic Rays and Particle Physics*, Cambridge University Press, 1990, ca. 280 pgs.

All of the people doing the typing were familiar with L<sup>A</sup>T<sub>E</sub>X/T<sub>E</sub>X, but were not familiar enough to modify and/or create macros to do what was necessary to meet the requirements of the publishers. As a result, most of this work was done by me. It is important to provide this level of support initially, because the pay-back on future books is invaluable with respect to time. Discoveries to share are:

- using the L<sup>A</sup>T<sub>E</sub>X book style required the least amount of work. This definitely depends on the publisher; and
- you need to have a large version of T<sub>E</sub>X to produce books of this size. The main problems have been cross referencing, size of captions in figures and tables, and size of tables. Our local configuration of T<sub>E</sub>X is listed below.

Parameter	Maximum
strings	4613
string characters	64042
words of memory	262141
multiletter control sequences	9500
words of font info	72000
fonts	255
hyphenation exceptions	607
stack positions	300i 40n 60p 2000b 4000s

With the above configuration I never run into a problem of T<sub>E</sub>X's capacity being exceeded, except when an actual error occurs. (You all know the case of the infamous missing }).

## Experience with Publisher-Defined Macros

Most of my experience has been with users who were supplied macros for journal publications. Each of these journals provided incentives for users to submit papers using its L<sup>A</sup>T<sub>E</sub>X/T<sub>E</sub>X macros.

1. American Geophysical Union, *Journal of Geophysical Research*

- saves time;
- final product looks better; and
- saves money on page charges (\$40 vs. \$140).

2. SIAM (The Society for Industrial and Applied Mathematics)

- provides greater control over the final appearance;
- eliminates introduction of errors from re-typing;
- eliminates one round of proofreading; and
- the author receives 100 free reprints.

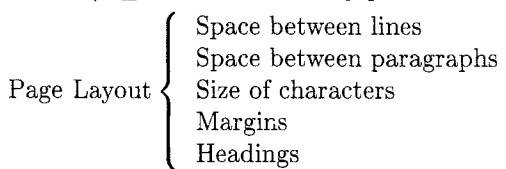
I suspect there are many other publishers who provide this service. It is my hope that more publishers will supply macros and documentation on how to meet their specifications and that this information will be published in *TUGboat* or some type of a newsletter to keep the T<sub>E</sub>X community informed. Regardless, it is obvious to me that the users of these L<sup>A</sup>T<sub>E</sub>X/T<sub>E</sub>X macros need to know more than just the basics. My concern is: How much more?

## L<sup>A</sup>T<sub>E</sub>X/T<sub>E</sub>X User as Typesetter?

Many users become frustrated because most of the time they don't need to be concerned with such details. They feel they get caught up in the details of learning L<sup>A</sup>T<sub>E</sub>X/T<sub>E</sub>X rather than in the actual writing. This was my primary motivation for setting up the thesis macros. I thought that it would be easier in the long run to have L<sup>A</sup>T<sub>E</sub>X/T<sub>E</sub>X users use my macros instead of designing new ones themselves. As a L<sup>A</sup>T<sub>E</sub>X/T<sub>E</sub>X consultant, I try to provide as much help as possible in meeting specifications for situations such as journals, books, etc. I quickly found out that many questions had nothing specifically to do with L<sup>A</sup>T<sub>E</sub>X/T<sub>E</sub>X, but in fact were questions about how to interpret publisher's specifications. These questions show that the users know

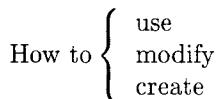
very little about the tools that exist to help ease the process of creating a document. As a result, I began to see that the following concepts can help most users ease their frustration.

1. Bridge the gap between the terminology used in L<sup>A</sup>T<sub>E</sub>X/T<sub>E</sub>X and that used by publishers.



I think that many L<sup>A</sup>T<sub>E</sub>X/T<sub>E</sub>X users (myself included) would be greatful for a document that lists the common terminology used by publishers and the proper L<sup>A</sup>T<sub>E</sub>X/T<sub>E</sub>X macros that correspond to each. Again, having this information published in *TUGboat* or some type of a newsletter would be an invaluable reference for the T<sub>E</sub>X community.

2. Understand the macros.



One comment: Don't reinvent the wheel! I spend a considerable amount of time finding out whether or not what I need has already been created or is close enough that I can modify it to do what I want. This is especially important for the casual user. It is much easier for the casual user to learn how to use a macro or change it slightly than to start from scratch. Most users are more than willing to solve the problem using this strategy and only resort to my help if they can't find a macro to do what they want. In fact I encourage users to call me before they get too frustrated. I'm glad they try themselves and I am all for self-sufficiency, but I don't like anyone to become so frustrated they want to give up totally.

3. Know the tools that make L<sup>A</sup>T<sub>E</sub>X/T<sub>E</sub>X easier.

- utilities for matching { and }, and \begin{...} and \end{...};
- spell checker (removing all control characters);
- screen previewing; and
- including graphics through PostScript

Many of these tools exist for different environments. Here are some that I find extremely helpful:

- Matching

**texmatch** is a program that checks matching in T<sub>E</sub>X and L<sup>A</sup>T<sub>E</sub>X documents.

It gives error messages if it detects unmatched delimiters. Delimiters are braces, brackets, parentheses, dollar signs (single and double), and L<sup>A</sup>T<sub>E</sub>X's \begin{ and \end. I know that this program is available on UNIX and PC systems.

- Spell check

**detex** is a filter that strips T<sub>E</sub>X and L<sup>A</sup>T<sub>E</sub>X commands from a file. This really helps in a UNIX environment before using a program like **spell** or **ispell**.

In our PC environment, most users prefer WordPerfect. Since WordPerfect contains a spell checker, all you need to do is to set-up the spell checker once to ignore the control sequences. This has a hidden advantage. Not only are all of the misspelled words caught, but also misspelled control sequences are found, thus avoiding a T<sub>E</sub>X error.

- Preview

The important point is not what package you are using, but that you have the capability to preview. In my opinion, one should not even consider creating a document that is going to be published without the ability to preview. So much time and paper is wasted without this tool.

- Graphics

Again, if something works for you, then more power to you! However, I have found that PostScript-capable printers are the most flexible and provide the best situation for incorporating graphics into a L<sup>A</sup>T<sub>E</sub>X/T<sub>E</sub>X document.

**dvips** is a program that converts a T<sub>E</sub>X .dvi file to a PostScript file.

**psfig** is a T<sub>E</sub>X macro package that facilitates the inclusion of arbitrary PostScript figures into T<sub>E</sub>X and L<sup>A</sup>T<sub>E</sub>X documents.

**macps** is a program that adds the appropriate Macintosh LaserPrep file to the beginning of a Macintosh PostScript file. This is very useful as a first step in printing a Macintosh PostScript file on a computer system other than a Macintosh. Other steps are required to include such a file into L<sup>A</sup>T<sub>E</sub>X/T<sub>E</sub>X documents. I know that this program is available for UNIX systems.

Once again, I hope a list of tools for L<sup>A</sup>T<sub>E</sub>X/T<sub>E</sub>X will be published with periodical updates in *TUGboat*, or some type of a newsletter that will contain all the information about what each tool can

do, what environment was the tool designed for and where can you find it. Right now, some very good information, such as "Frequently Asked Questions about *TEX*" and "Supplementary *TEX* Information (FTP sites)" is available. But is there more that I am not aware of?

## Conclusion

How does all of this impact an organization?

1. The user must invest a considerable amount of time learning how to produce a document based on the publisher's specifications. Here is where a considerable amount of time can be saved if the publisher supplies the necessary macros. It is also important that users who do this type of work be recognized for their skills as typesetters rather than as typists.
2. Efforts must be made to offer good support to users so the documents can be completed in a timely fashion. Support has been the key to users being willing to use *TEX* and/or *LATEX* for publishing documents. Many times I thought to myself, "Why did I ever suggest using *TEX* or *LATEX*?" My reason is obvious when the final document is printed: Nothing compares to the quality of *TEX*.
3. Using *TEX* can save dollars. The final cost of a book or the cost of publishing a paper is certainly going to save an organization money. However, there are hidden costs that should be kept in mind. The user now spends more time inputting the material and taking care to format the document correctly, and I spend more time helping people who need to follow a specific layout.

## Update

During the TUG conference, there were a few very interesting points that surfaced and I feel they should be included in the paper.

1. There are many publishers who are interested in authors as compositors. In talking to many of these publishers, I found that most of them are making efforts to provide style files for *LATEX* and/or macros for *TEX*. This was very encouraging to me. I believe that a list of publishers and the style files/macros they supply and/or accept should be published in *TUGboat* or some newsletter to the *TEX* community.
2. There is a big difference between a graduate student writing a thesis/dissertation and an author of a book. A graduate student can be threatened with not graduating if he or she does

not follow the specifications, but what about authors? Can a publisher threaten to not publish the book? I don't think so. It is important to both parties that the book is published, but compromise is essential. I feel that authors have to realize that publishers are the experts in designing books and the author is the expert about the content of the book. Comments:

- If an author is planning on using *LATEX/TEX* then choose a publisher that accepts this format or, more importantly, one that has experience using this format.
- An author should discuss design specifications with the publisher as early as possible and decide what is acceptable.
- The publisher needs to specify clearly what types of changes to the design specification are acceptable in order for authors to express themselves.

Lack of communication between the authors and publishers seems to be the number one problem. Speaking from a support point of view, making these issues as clear as possible up front will save everyone concerned time and energy.

3. There may be no savings, or an actual increase in cost can occur when using *LATEX/TEX*. I was very surprised to hear that this can happen. In many cases, costs stayed the same or increased because authors insisted on certain design changes. Here is a primary example of where my comments in 2. can help.

## Bibliography

- Academic Computing Services. *The LATEX UD Thesis Format*. University of Delaware, Newark, June 1990.
- Academic Computing Services. *The TEX UD Thesis Format*. University of Delaware, Newark, June 1990.
- Knuth, Donald E. *The TEXbook. Computers and Typesetting*, Vol. A. Reading, Mass.: Addison-Wesley, 1986.
- Lamport, Leslie. *LATEX: A Document Preparation System*. Reading, Mass.: Addison-Wesley, 1986.

# Simultaneous Electronic and Paper Publication

John Lavagnino

Department of English and American Literature, Brandeis University, Waltham, MA 02254 USA  
617-736-2080  
Internet: [lav@binah.cc.brandeis.edu](mailto:lav@binah.cc.brandeis.edu)

## Abstract

Many current applications in computerized text processing involve the creation of “multiform texts”. Such a text is designed for use in several forms: in both print and electronic form, for example. This is a valuable goal for many kinds of text; one example that may perhaps seem unlikely is the edition in progress of Thomas Middleton’s complete works. The central question in creating a multiform text is the choice of a language for the basic text files; SGML seems to be the best choice. It has worked well on the Middleton project so far, and has worked well together with *T<sub>E</sub>X* in solving some of the problems that have arisen specific to this text.

## Multiform Text

An underlying thread connects a number of different projects in the computer processing of texts: the idea of a “multiform text”, a work that is meant to be read and used in several different forms—most characteristically, both electronic and printed forms.

One of the most familiar instances of such a text is the computer manual. If you’re writing a manual for a computer you’re likely to be using a computer to make it; and then why not use the computer to access it as well? That access doesn’t necessarily require giving any special thought to making the electronic edition useful: many of us have long depended on having a copy on disk of the *T<sub>E</sub>X* source for *The T<sub>E</sub>Xbook*. A text editor is all you need to work with it.

But one does not read *T<sub>E</sub>X* source very happily: this system is fine if you want to look at a macro definition, but it’s unsatisfactory if you’re interested in what an example produces on the printed page. More interesting are those systems that attempt to provide both print and electronic versions that are equally usable. On-line help systems for computers often work from a textual base that’s adapted from, or also issued as, print manuals: both usually contain much the same information, and the attraction of writing the documentation once, not twice, is obvious. The UNIX *man* command is one familiar example: it draws on text encoded in the *troff* typesetting language, and formats that text for display either on the user’s terminal or on a printer, so that when you ask for help on a

command, you get the same text that’s presented in the printed manual.

This system is possible because the documentation is encoded in a way that doesn’t make it impossible to print on a typewriter-like device instead of on a real typesetter. The on-line access, however, gives you nothing more than page images; these provide as much information as the printed manual, but they also provide *no more* than that. In contrast, the programs distributed by the Free Software Foundation use a more sophisticated documentation system that takes better advantage of the computer’s powers for structuring text in ways not available in print. The Texinfo system not only uses a descriptive markup (based on *T<sub>E</sub>X*) that encodes the structure of the text and lets macros decide how to present the information; it also includes encoding for cross-references that allow a user who’s got the GNU Emacs editor to more effectively find information in the manual and move to related topics in it (see Stallman and Chassell).

Such a system combines the advantages of print and electronic editions. The print user can still read in bed, write on the copy, suffer less eyestrain, and use the document when the computer is down; the electronic user can search for a topic or phrase much faster, follow connections that may not be represented in the sequential text of the printed manual, and get assistance with a program from within that program. The best use of the system seems to come not from using one or the other form exclusively, but from switching back and forth,

using the form which is best for addressing each momentary need.

That sort of combined publication needn't be limited to computer manuals. The advantages of multiform text are there for all sorts of works.

## The Oxford Middleton

My own interest in the multiform text comes out of my work as one editor of an edition of the complete works of Thomas Middleton (1580–1627), the English Renaissance playwright. This edition is being prepared by an international team of editors for publication by Oxford University Press in 1994. It will include the texts of all Middleton's works—above all, his twenty-seven extant plays, but also numerous masques, entertainments, poems, and prose works; and it will provide introductions and very detailed notes to all these works. This is the first complete edition of Middleton's works in over a century, and we hope that it will not only collect all the accumulated scholarship on Middleton, but also establish his importance as a writer.

Such a complex work is usually quite expensive to set in type, so that the advantages to us of using *T<sub>E</sub>X* to do the typesetting ourselves are clear. The advantages of creating a multiform text (instead of concerning ourselves solely with entering the right *T<sub>E</sub>X* codes to print the work) may be less immediately apparent. Yet a multiform text is of value both for us, during our preparation of the work, and for other readers and scholars after its publication. Editors and scholars have long depended on concordances to help them in understanding the characteristics of an author's style and thematic concerns; the electronic text gives us, in effect, such a concordance to the text as we prepare it, rather than long after it's published. Providing an electronic text also makes possible a later conversion of the work into a hypertext that can allow readers quick access to the various sorts of notes to the work.

The creation of a multiform text is not an experimental approach, but instead one that keeps the labor for everyone to a minimum and creates the most valuable print and electronic editions. In the following discussion of the salient issues in this case, I'll mention these work requirements as they come up.

## The Choice of a Language

Most of the important questions about how to create a multiform text are related to the choice of

a “markup language”—the language in which the text and its structure are specified. The basic idea is to choose one form for the text from which all other forms, electronic and printed, will be derived. The markup language for this basic form should make the derivation of other forms work easily and well.

The UNIX *man* command, and the GNU Texinfo system, both use typesetting languages with macro capabilities—*T<sub>E</sub>X* itself, in the latter case. And that choice might seem to make sense in the general case: after all, one thing we want to make is a printed text created by *T<sub>E</sub>X*, and so using *T<sub>E</sub>X* as our markup language seems to automatically solve the problem of creating one of our final forms. But it isn't a helpful choice when it comes to the electronic side: *T<sub>E</sub>X* is not especially easy to translate into other markup languages. The nature of its macro definition facilities means that a program needs to know rather a lot of what *T<sub>E</sub>X* knows if it's to be able to make the conversion. Consider the rules in *T<sub>E</sub>X* for determining when a macro name has ended: according to Knuth [page 47], these require that we know the difference between letters and other categories of characters—a distinction that can be changed by a *T<sub>E</sub>X* input file. Argument delimitation is still more complicated [Knuth, pages 203–204]. (I am assuming here that the most desirable approach is to transform the basic form directly into other electronic forms. Carr and Partl have discussed separately approaches based on taking *dvi* output and converting it to other electronic forms, approaches that make things still more difficult.)

For the Middleton edition, we have chosen SGML, and use *T<sub>E</sub>X* only for the typesetting, not for our text representation. (See Laan for an introduction to SGML.) SGML is, first of all, rather easy to convert to other forms: the names of “tags” and “entities” in SGML, two different sorts of commands that are similar to different aspects of *T<sub>E</sub>X* macros, are in the normal usage terminated in an unvarying way—by ‘>’ for tags, and ‘;’ for entities. Converting our text from SGML to *T<sub>E</sub>X* seems to require nothing more complicated than global substitutions, and a few simple *T<sub>E</sub>X* macros to deal with the product.

A more important reason for choosing SGML lies in another facility it offers. It is intended not only to handle the electronic representation of a document's structure, but to allow the specification of rules governing that structure, and verification of a document's conformance to that structure. *T<sub>E</sub>X* checks only that you aren't transgressing the rules

of its input syntax; it has no facility for ensuring conformance to any specifications narrower than those in the *The T<sub>E</sub>Xbook*. One can build such specifications into any macro set, to some extent: L<sup>A</sup>T<sub>E</sub>X provides an example, in its checks on the proper nesting of \begin{ and \end commands (among other things). L<sup>A</sup>T<sub>E</sub>X still doesn't check everything, and its specifications are those of the whole macro set, not a user's subset.

This question of verification matters for any kind of text, but it's of particular importance with the multiform text. It's necessary with such a text to keep close tabs on what commands are used: you want to ensure that you don't wind up with something that can be represented in one medium but not the other. We're familiar with struggles to get a page printed just right; but there's another level to the problem in this perspective, that of getting it "just right" in more than one medium. SGML helps prevent surprises in this realm.

The use of SGML's facilities does require some extra work to formulate the specifications for the text's structure, but some consideration of those specifications has usually been necessary with multiform texts; the advantage of SGML is that it can help to enforce those specifications.

SGML is also more securely oriented than any typesetting language towards encoding the structure and meaning of text elements, and not the details of how they're to be printed. The importance of an encoding that is focused on structure and meaning has already been argued at length (see Coombs et al. for a theoretical presentation, and Lafrenz for a publisher's agreement with it on practical grounds). Greater abstraction will also help us with uses we've never anticipated (but which may suddenly be of importance when our publication date in 1994 rolls around): our ability to generate new forms of our text will be enhanced if we have precise specifications of what's going on in our text.

Finally, for our particular project, there is the advantage that the International Text Encoding Initiative is currently developing guidelines based on SGML for tagging electronic texts, with particular attention to the needs of scholars (see Sperberg-McQueen and Burnard for a draft of these guidelines). We want our text to get used and studied, and adhering to standards is a good way of doing that.

Most people get the impression that SGML is not good to use for data entry, because its markup appears to be very bulky. This is only true when no use is made of the extensive provisions SGML makes for minimizing the markup; with proper use of these

features, SGML requires no more typing than T<sub>E</sub>X does. But we began our project without any SGML tools, and so we handle the data entry in another way. Our approach has been to devise a very terse markup that's used solely for data entry, adapted very narrowly to the kinds of texts we're encoding; we convert this immediately to SGML, and perform all further processing on the SGML files. The creation of the programs to do this conversion has been one of the principal tasks involved in setting up this mode of working — though it has hardly been an onerous one. If we had obtained appropriate SGML tools at the beginning of our work, even this task would have been unnecessary.

## Referring to the Printed Text

The careful choice of a markup language should make it possible to contain the problems that come from our need to do a great deal of computer processing of our text; it should make the necessary transformations easy, and ensure that we aren't entering textual elements that can't be processed within both realms.

But there is another layer of problems that can arise. What would happen if we needed to include information in an electronic text about the details of how the printed text looked? That would mean that the printed text would not just be a spinoff of the electronic text, but that we'd need to extract information from our printed text — or from the *dvi* file — and fold it back into the electronic version; it could be a difficult task.

The conventional index is a good example of this: the text of an index is an analysis of the book in which it appears, and it's dependent in a very sensitive way upon how the page makeup came out. Of course, we know how to handle index-making with T<sub>E</sub>X. Its \write command is designed to facilitate capturing information for an index or table of contents that needs to know about page numbers. In other multiform texts it's been common to use references not to page numbers but to important structural divisions, which don't depend on the page makeup: the UNIX documentation that's used by the *man* command is broken up into small chapters, rarely more than a few pages long, one chapter for each command.

The particular traditions of publishing in literary studies pose a problem for us with Middleton. One demand that scholars make and are not going to give up is for a very precise system for referring to particular lines in the text, a system traditionally

implemented, in fact, with line numbering. Middleton's plays are typically written in a mixture of prose and verse, often changing within a speech. Prose is traditionally numbered using physical line numbers: that is, each actual line of type is counted as a line. Verse is numbered using logical or structural line numbers: a line of verse may take more than one line of type to print, but it's still counted as only one line in this numbering scheme. On top of this, stage directions are handled in a different way: whenever a stage direction appears that's not on the same line as spoken text, it's given a physical line number in a decimal numbering annexed to the previous speech's line number: 18.1, 18.2, etc. The stage directions that appear at the opening of a scene are numbered 0.1, 0.2, etc.

The force of tradition makes it impossible to use a different system (and it seems difficult to come up with another that would be as precise and as easy to use for readers of the printed text). We can print such line numbers readily enough, using the EDMAC macros (see Lavagnino and Wujastyk). EDMAC can also create footnotes and endnotes that use such line numbers in references, but we also need to get them back into our SGML text: that is, to mark in the SGML text the point at which each line begins. The reason is that users of the electronic version, as well as users of the paper version, need to be able to look things up using these line references, and to find the line address for a passage so that they can tell others where they're looking.

This is a problem because the line numbers that we ultimately want to fold back into our base text are all generated in the course of typesetting, and actually it's not an easy matter to find out what they are and get them back into our SGML text. Consequently, there's a need for software that can take information out of our typesetter file—out of a file that is usually deliberately made to focus on niggling presentational details and tell us nothing about structure—and interpret it for incorporation into the SGML. It's a striking instance of how the printed page is not merely an end product that leads no further, at least not within the electronic world.

I said that one reason behind our use of SGML was to stress the representation of meaning rather than structure in our text. But the reference-system problem leads to a curious inversion of this situation: if we want a print-based reference system, we must process the output from our text formatter—output which consists of text that's been converted to a format that tells us as little

as possible about *meaning*, and far too much about appearance.

For ordinary prose this isn't really a huge problem. In dramatic texts, line numbering is complicated, being partly logical and partly physical. It's quite difficult for a program to determine the numbers by just looking at the type on the final page, unless every single line is numbered. It can be done, but at the expense of writing a program that's highly dependent on the details of how your pages are laid out, since a lot of the clues that we as readers depend on to figure out whether something is prose or verse or whatever have to do with indentations, details of spacing, and font selection.

Our approach to this problem puts all the burden of assigning line numbers to blocks of text on *TEX* itself. Rather than try and write software that guesses the line numbers, we have *TEX* itself issue \special commands at the start and end of every line of text: to specify the line number, and to mark that text as a part that is numbered (since every page includes headings, marginal line numbers, and other text that is part of the presentation of the text, not the text itself). This much is a simple extension of the EDMAC macros that generate the line numbers: those macros already add each line of text to the output page separately, so inserting the \special commands that enclose each line of text is straightforward.

The bigger task is interpreting the resulting *dvi* file: we need to convert it into something that's close enough to our original SGML file that we can match up the texts and see where to put the line numbers. This is by far the most substantial programming task that the production of the Middleton edition has required so far, and I don't expect that anything to come will prove more difficult. The problem, however, would be more difficult with any typesetter other than *TEX*. Not only does it have the unusual, but very useful, \special mechanism; it also comes with its binary-file formats documented in a very thorough manner, and with ancillary programs that already demonstrate how to read things like *dvi* files. Indeed, the *dvitype* program already does a great deal of the task for us: our preliminary version of this software simply starts from *dvitype* output, not from the *dvi* file itself.

Although this is a thorny problem, it is one whose solution is made much simpler by certain well-known (but perhaps insufficiently-appreciated) merits of *TEX*: its extensibility, its excellent documentation on its internal workings and file formats, and its wealth of supporting programs, all available in source code.

## Conclusion

Many people who prepare texts on computers are already finding themselves drawn to the creation of multiform texts. In this account I've tried to identify the general questions that should be considered in doing this; but it's probable that others will also run into problems specific to the kind of text they're working with, as we have with reference systems in our work. The use of flexible tools also makes the successful resolution of these problems easier.

## Bibliography

- Bell Telephone Laboratories. *Unix Time-Sharing System: Unix Programmer's Manual*. Seventh edition. Murray Hill, New Jersey, January, 1979.
- Carr, L., S. Rahtz, and W. Hall. "Experiments in *TEX* and HyperActivity." *TUGboat* 12(1), pages 13–20, 1991.
- Coombs, James H., Allen H. Renear, and Steven J. DeRose. "Markup Systems and the Future of Scholarly Text Processing." *Communications of the ACM* 30(11), pages 933–947, 1987.
- Knuth, Donald E. *The T<sub>E</sub>Xbook*. Reading, Mass.: Addison-Wesley, 1984.
- Laan, C. G. van der. "SGML (, *TEX* and ...)." *TUGboat* 12(1), pages 90–104, 1991.
- Lafrenz, Mimi L. "Textbook Publishing — 1990 and Beyond." *TUGboat* 11(3), pages 413–416, 1990.
- Lavagnino, John, and Dominik Wujastyk. "An Overview of EDMAC: A plain *TEX* Format for Critical Editions." *TUGboat* 11(4), pages 623–643, 1990.
- Partl, Hubert. "Producing On-Line Information Files with L<sup>A</sup>T<sub>E</sub>X." *TUGboat* 10(2), pages 241–244, 1989.
- Sperberg-McQueen, C. M., and Lou Burnard, eds. *Text Encoding Initiative: Guidelines For the Encoding and Interchange of Machine-Readable Texts*. July 1990.
- Stallman, Richard M., and Robert J. Chassell. *Texinfo: The GNU Documentation Format*. Cambridge, Mass.: Free Software Foundation, 1988.

# SGML versus/and $\text{\TeX}$

Robert W. McGaffey

Oak Ridge National Laboratory, Building 2506 MS 6302, P.O. Box 2008, Oak Ridge, TN 37831-6302 USA  
615-574-0618; FAX: 615-574-1001  
Internet: [mcgaffeyrw@ornl.gov](mailto:mcgaffeyrw@ornl.gov)

## Abstract

Everyone who handles computer documentation faces the problem of proliferating application-specific versions of a source file and the added difficulty of merging changes back into the source. SGML is a resource for building a generalized solution.  $\text{\TeX}$  and SGML offer a particularly harmonious synergism for documentation applications.

## Consider This Problem:

You have a database with important information. You need to publish some of the information and wrap it inside appropriate text. Furthermore, you need to create an abstract for printing inside a professional journal in preparation for a presentation of your paper at an international meeting. Meanwhile, a colleague calls and requests a copy of some of the database for his/her research *if* you furnish it. Finally, your latest experiment dictates that you must change some of the data you have already placed in half a dozen places. Wouldn't it be grand if you could just keep all of that information in one place and only have to modify one copy and be sure that all of your data was up to date? Of course it would. But you can't. Well, suppose you could keep one "official" copy and automatically generate all of the others whenever the "official" copy changed? Would you be interested? If so, welcome to the world of SGML.

## What Is SGML?

SGML (Standard Generalized Markup Language) is an international standard which purports to standardize the way information is marked up in a storage medium. But in practical terms, SGML implies a system of programs which helps us to create both the "official" computer file and the automatic copies we have to generate. Here's how it works:

First we create a file, called a Document Type Definition (DTD), which describes completely how the information is organized in the "official" file. The DTD is intended to support hundreds or even thousands of documents organized in the same way.

For example, consider a "theme" which will contain one title followed by one or more paragraphs.

Next, we can build what is called an "instance" file, which is the official name for our "official" file. A simple DTD file could lead us to the following instance file:

```
<document>
<title>The title</title>
<p>The first paragraph.</p>
<p>The second paragraph.</p>
</document>
```

Note that a SGML "element" is most often represented by something like `<tag>(the SGML element named "tag" goes here)</tag>`.

But our sample is not what I call *real* SGML because it is so attached to the formatting of the theme. We should be happier to see something like:

```
<theme>
<title>The title</title>
<idea>The first paragraph.</idea>
<idea>The second paragraph.</idea>
</theme>
```

Note that we are now tagging information rather than the formatting of the information.

Smart editors already exist which will help us with the two steps above. They will help us create a legal DTD file and then make sure that the instance file we create matches the DTD we created.

Now, we have our "official" file. How do we get the automatic copies?

Other programs exist which parse the instance file and translate it into another format. That means that I can translate my "official" file into my favorite typesetting language ( $\text{\TeX}$ ) by writing a program for my parser-translator and when I do so I will get something like:

```
\starttheme
\starttitle The title\finishtitle
\startidea The first paragraph.\finishidea
\startidea The second paragraph.\finishidea
\finishtheme
```

which, with a suitable set of *TEX* macros, will generate my paper copy.

## Why Is SGML Better Than *TEX*?

Because it is more than a typesetter. Consider the way in which *TEX* typesets a superscript. How does *TEX* indicate a footnote marker?, an atomic weight?, the degree symbol?, and sometimes trademarks etc.? Answer: often all by the same mechanism; i.e.,  $\hat{(whatever)}$ . Real SGML forces you to treat these differently because they have different meanings. Thus, an SGML document has more inherent intelligence than a *TEX* document.

Since SGML is independent of the programs used to typeset, store in a database, extract an abstract, etc., your documents become "official", i.e., the one and only storage medium needed to hold all of the information. Therefore, any typesetter, database, etc., may be accommodated by changing only the program which drives your parser-translator. Thus a change in one file makes the new output automatic. So if I decide to switch typesetters from *TEX* to *NIT* without modifying my instance files at all.

The first time I heard that, my objection was that instead of having *TEX* files you then have SGML files and, "What happens if SGML changes and I want to convert my files to NISGML?" what is the difference? The answer is, if SGML is ever modified (and I sincerely hope it is!), all of your SGML files could be run through the parser mentioned earlier and converted in one huge batch file. Try doing that with any other system and you will get an appreciation for SGML and the available parsers.

## Why Is *TEX* Better Than SGML?

When you consider the example of a mathematical expression, taking full advantage of all the capabilities that SGML offers, you cannot read the equation. The corresponding *TEX* equation is rather easy to read. The SGML document may also contain cumbersome structures necessary to distinguish between the various uses of say, periods. In such cases, the *<filename>.tex* document is much easier to type and to read than is the instance file. But these are excuses—if the information needs to

be available, then the tagging needs to be done no matter what the result is in the instance file.

Naturally, *TEX* is better at typesetting because SGML is not a typesetting system. There are those who try to make it so by misusing SGML's attributes and forcing SGML files to contain formatting information. But it is not. It should be used to mark the information without regard to its format. There are even those who would sacrifice printing quality for the sake of the instance file. But when you can have your SGML file and *TEX* it too, why not?

## Then We Say, "SGML and *TEX* for Publishing"

There are at least four main reasons for inserting SGML in front of *TEX*:

1. Due to the writing of smart editors, it is much easier to create a properly structured document with SGML. But what we did not mention before is that the smart editors *force* typists to enter all of the data and in the correct order. (It is true that you can still make a mistake but you have to make it on purpose.)
2. SGML allows us to have the luxury of the "official" document. I don't think that I will appreciate all of the problems that this solves until we have had a working system in use for some period of time.
3. SGML can be used (with the proper parser-translator) to generate input to a database system as well as other typesetting systems or anything else where information is stored. Thus, the input information can be confined to one file (the instance file) and yet many output mediums are possible.
4. Because of extra intelligence available from SGML, the *TEX* macros needed to typeset the document are easier to write. Consider, for example, that we do not need to concern ourselves with whether or not our document has (or does not have) a title because we will know that it does. Thus, our *TEX* macros need not check for this.

There are at least four reasons why *TEX* should follow SGML:

1. There are SGML aficionados who feel that typesetters like *TEX* should be executed quietly behind the translator and that users need never know that it is being used at all. *TEX* is one

of the most programmable typesetters around and thus more capable of this than other typesetters. For example, consider typesetting a table from an instance file. SGML is aware of the structure of the information but has no idea of the lengths of the various elements. So, typesetting a table from an instance file means adopting some standard format which

will hopefully satisfy some high percentage of tables. Some of the table information will be wide and some narrow. And it is not likely that the lengths of the table headings will correspond to the lengths of the data they head. As a result, the simple table of Figure 1. becomes "strung out" and ugly because its headings are so long.

	No. of samples	Exposure rate			Standard Error
		Max	Min	Ave	
Input Summary	80	110	6.6	18	3.5
Output Summary	254	11	5.8	7.1	0.043

Figure 1: Table generated with eyes closed.

	No. of samples	Exposure rate			Standard Error
		Max	Min	Ave	
Input Summary	80	110	6.6	18	3.5
Output Summary	254	11	5.8	7.1	0.043

Figure 2: Table generated after seeing disaster above.

Most of us would probably agree that the rendering shown in Figure 2 is better. *TEX* is one of few typesetters capable of making the decision to change the pattern of the table on the fly. That is, *TEX* is smart enough to decide that the second alternative is better, all by itself. The number of places where such decisions could be made is probably only a limit of our own imaginations. Thus, *TEX* can hide behind SGML better than most typesetters. But there are cases that cannot be handled by *TEX* automatically.

2. There are things that you can do with *TEX* that make the published-on-paper copy so clean. For example, *TEX* can assure us that all of the columns of any output page will never

end in a hyphenated word and yet will be the same length. The author thinks the best way to do this is to look at the final document and then use `\hbox{(hyphenated word)}` whenever it is needed. Sometimes the use of `\looseness=(number)` is also required to prevent widows and orphans in such columns.

3. Editors will want to make formatting changes. Editors *always* want to make changes.
4. Yet the main reason to use *TEX* as a backend to SGML is the reason we all started using it in the first place. It is still true that nothing else sets math like *TEX* (or paragraphs either) so we should continue to use *TEX* simply because it does such a beautiful job.

# Typesetting SGML Documents Using TEX

Andrew E. Dobrowolski  
ArborText, Inc.  
internet: [aed@arbortext.com](mailto:aed@arbortext.com)

## Abstract

Since its publication as an international standard in 1986, the Standard Generalized Markup Language (SGML) has become a preferred document-markup standard within many industries. Many users have developed their own document type definitions (DTDs) that define the elements (tag sets) for their documents. However, if SGML is to become a universally accepted standard of document interchange, then a standard way of specifying formatted output and a means of producing that output will be needed.

The U.S. government's Computer-aided Acquisition and Logistic Support (CALS) initiative selected SGML as the standard for text interchange. The output specification section of the CALS standards proposed the Formatted Output Specification Instance (FOSI) as the means of formatted output specification interchange.

TEX can be used as the formatting engine to implement FOSI-based formatting. But without extending TEX, not every FOSI formatting request can be fulfilled. Conversely, certain TEX capabilities cannot be formulated in terms of FOSI characteristics. However, a FOSI/TEX-based formatting system would be a major advance towards fulfilling the document interchange needs of a growing community of SGML users.

## Document Interchange Standards

In the past ten years, TEX has become a well known and widespread language for typesetting technical documents. From its original base of universities and colleges, it has spread to such an extent that people in industries with only incidental needs for publishing have heard about it. A large part of TEX's appeal comes from its portability, since the program is in the public domain and has been ported to quite a number of operating systems. There is no standard for the way a TEX document is "marked up"; this is dependent on the macro package used. Given the right macro package and fonts, the formatted output of two different TEX implementations on two different machines will produce identical results.

By contrast, generic markup systems identify document structures without making assumptions about the end application of the document. This makes the same document useful to various programs and for various applications. Generic markup has been around in several flavors for over ten years.

These dissimilar flavors were a hindrance to its utility. To remove this hindrance and to promote the portability and acceptance of generic markup, an international standard (IS) specification for generic markup was established in 1986. Since then, SGML (Standard Generalized Markup Language) has become extremely important to industry, especially in areas where huge quantities of data have created a document-management nightmare. Today a large number of programs can read and write SGML on a variety of platforms.

The U.S. government's Computer-aided Acquisition and Logistic Support (CALS) initiative gave SGML additional clout by selecting SGML as the standard of text interchange between the Department of Defense and its subcontractors. However, SGML contains no information pertaining to the printed representation of a document or to the meaning attached to the markup. The companion standard to SGML that addresses standardized formatting specifications, the Document Style Semantics and Specification Language (DSSSL), is

still in the design stages. It is not expected to become an international standard until at least 1993. For this reason the output specification section of the CALS standards proposed the Formatted Output Specification Instance (FOSI) as the means of output specification interchange.

## SGML and FOSI Structure: An Overview

All SGML documents must conform to certain rules that are defined partially by the standard and partially by a prolog to the document; this prolog is called the document type definition (DTD). The DTD defines the "elements" of a document; in a document instance, these are marked off by start tags and end tags. For example, a hypothetical section might be marked up like the fragment in Listing 1. Here, `<head>` and `</head>` (pronounced "head" and "end head") are start and end tags that delimit the head element. The parent of head is section and its siblings are the two para elements.

A DTD also defines what "attributes" are associated with an element. An attribute is an annotation that appears in the document instance and augments the information provided by the markup. Attributes appear within an element's start tag. If the element "head" has an attribute "id" for use in cross references, then that attribute can be assigned some value in the document instance, for example: `<head id="overview">`.

It is important to note that SGML allows the same element to appear in many contexts within a document structure. The same markup can be used to describe a chapter head, a section head, and even a table head. At some point, a distinction must be

---

```
<section>
  <head>SGML and FOSI Structure:
  An Overview</head>

  <para>All SGML documents must conform to
  certain rules that are defined partially by
  the standard and partially by a prolog to the
  document, which is called the document type
  definition (DTD).</para>

  <para> In addition to being first off the
  starting blocks to becoming a national
  standard, the FOSI is also the most
  manageable. </para>
</section>
```

---

**Listing 1.** A Document Instance Fragment.

made between these various contexts, at least for the purpose of formatting the document. But since the DTD also restricts the context in which any element may appear, the task of defining the style of every element in every one of its possible contexts is fairly well defined. Thus, a FOSI will not define the formatted output style of a document element but of an element in context (or e-i-c).

Many industries have developed DTDs that define the elements (tag sets) used to mark up their documents. Before SGML becomes a universally accepted standard of document interchange, one of SGML's companion standards for output specification must be fully implemented. TEX could be the engine in the implementation, the means of producing standardized output for any SGML document. The ultimate goal would be to make this process automatic for the arbitrary DTD document. The only information that would need to pass from one site to another in order to print a document would be the document instance, the DTD, and an output specification.

It appears that of all proposed output specification standards, the FOSI is the closest to becoming a recognized standard. In addition, the FOSI specification is the easiest to implement. A FOSI is itself an SGML document that conforms to the Output Specification (OS, or outspec) DTD. But, instead of being made up of parts, chapters, or sections, a FOSI is made up of divisions that describe page models and the output format of each of the document's elements.

There are six major divisions in an output specification instance: the security description (secdesc), the page description (pagedesc), the element style description (styldesc), the table element style description (tabdesc), the graphical element description (grphdesc), and the footnote area description (ftndesc). All but the pagedesc and styldesc are optional. There still is no definition for the output style of mathematical formula elements. Thus, the mathematics must either be passed through in the native language of the formatting system and translated into the native language by the translator, or the output specification for the mathematical elements must be "hard wired" in the formatting system.

The style description is the most important division of the outspec for simple text documents. The styldesc contains a document description (docdesc), zero or more environment descriptions (envdesc), and at least one formatting specification for an e-i-c. It is in these subdivisions that special FOSI elements called categories appear. Each category

## SGML and FOSI Structure: An Overview

All SGML documents must conform to certain rules that are defined partially by the standard and partially by a prolog to the document, which is called the document type definition (DTD).

In addition to being first off the starting blocks to becoming a recognized standard, the FOSI is also the most manageable.

**Figure 1.** Typeset Document Fragment.

provides data on a different aspect of the formatted output. There are 24 categories (with names such as font, leading, etc.), and each of these has from one to 13 attributes. These, when fully specified, exactly define the formatting aspect with which their category is concerned. These attributes are called characteristics, of which there are 128 in total. Once values for all the characteristics of any given e-i-c have been determined, it should be possible to define the appearance of that e-i-c on the printed page.

The categories control the font, leading, hyphenation, word spacing, letter spacing, indents, horizontal justification, highlight, change marks, prespace, postspace, page breaking, vertical justification, text breaking, spanning, page borders, ruling, character fill, enumeration, print suppression, automatic generation of text, automatic generation of graphics, the saving of text for cross reference, and the use of text saved for cross reference.

As mentioned above, the elements that may appear in a styldesc are docdesc, envdesc, and e-i-c. The characteristics of the docdesc define the style of the overall document and specify the default values for characteristics that are needed but not specified in an e-i-c. When used in this way, the docdesc is called the default environment. The envdesc section defines "named" environments that may be used instead of the default environment. The actual style definition for an element in a particular context in the document instance is given by an e-i-c. The SGML terminology for an element's name is the generic identifier (gi). An e-i-c specifies an element, its context, and its occurrence within that context

```
<e-i-c gi="section">
  <charlist>
    <font family="cm">
      <presp nominal="30pt"
             minimum="30pt" maximum="30pt">
    </charlist>
</e-i-c>

<e-i-c gi="head" context="section">
  <charlist>
    <font inherit=1 style="sanserif"
        size="14pt" weight="bold">
    <leading lead="14pt">
    <quadding quad="right">
    <keeps keep="1" next="1">
    <postsp nominal="24pt"
              minimum="20pt" maximum="30pt">
  </charlist>
</e-i-c>

<e-i-c gi="para" occur="first">
  <charlist>
    <indent firstln="Opt">
  </charlist>
</e-i-c>

<e-i-c gi="para" occur="nonfirst">
  <charlist>
    <indent firstln="15pt">
    <presp nominal="6pt"
           minimum="4pt" maximum="6pt">
  </charlist>
</e-i-c>
```

**Listing 2.** FOSI fragment.

by using the gi, context, and occur attributes, as shown in Listing 2.

Furthermore, this FOSI also uses the occur attribute of an e-i-c to make a distinction between the output format of the first and non-first occurrences of the para element. The paragraph indent of the first para within a structure is zero, while non-first paragraphs have an indent of 15 points and an additional prespace of 6 points. Figure 1 shows the formatted output from the document instance fragment. Characteristics not explicitly listed in the e-i-c definitions default to the values specified in the docdesc (not shown).

## SGML-to-T<sub>E</sub>X Translation

As with most SGML documents, the FOSI must first be read by an SGML parser or a dedicated program, and then translated into a form suitable for the formatting engine. Likewise, the document instance

must be translated by some process into a suitable form.

Translating a FOSI into TeX creates a series of macro definitions that appear in the TeX translation of the document instance. Given a suitable starting set of macros, it is possible to load the new macro definitions produced automatically from the FOSI translation and to format the document.

Because the output specification for a given document element is context sensitive, either the translation process or TeX must track and differentiate between differing contexts. To make the work of the macro package easier, the context sensitivity should be built into the translation process. In fact, TeX's limited look-ahead capability dictates that the translation will be context sensitive. TeX cannot recognize when an element is the last of its kind within the parent structure, but some occurrence conditions require that this distinction be made. For example, the last item in a list may need to inhibit a page break from separating it from the second-to-last item. This occurrence recognition must therefore be done by the translation process.

The easiest way to accomplish this is to give each e-i-c in the FOSI a distinct name and to use that name, when appropriate, in the translation of the document instance. Listings 3 and 4 show the translation into TeX of the document instance from Listing 1 and the sample FOSI fragment of Listing 2. Notice how the two sets of <para>...</para> tags are translated according to their occurrence.

---

```
\section{}
\sectionhead{}SGML and FOSI Structure:  
An Overview\endsection{}

\firstrpara{}All SGML documents must  
conform to certain rules that  
are defined partially by the  
standard and partially by a  
prolog to the document, which is  
called the document type  
definition (DTD).\endfirstrpara{}

\nonfirstrpara{}In addition to being  
first off the starting blocks to  
becoming a recognized standard,  
the FOSI is also the most  
manageable. \endnonfirstrpara{}

\endsection{}
```

---

**Listing 3.** Translation of Document Fragment.

## Implicit Specification of Characteristics

Let us examine more closely the specification of the first para e-i-c in the FOSI fragment in Listing 2. It explicitly sets the values for the `firstln` characteristic of the "indent" category and the `startln` and `endln` characteristics of the "textbrk" category; however, it neglects to explicitly define many other important formatting parameters. Nowhere was the font mentioned, or the prespace, or the justification (quadding). Nonetheless, as the formatted output suggests, these characteristics are well defined. In general, one of two implicit methods is used to determine the value of a characteristic not mentioned explicitly in an e-i-c.

One of the methods is inheritance. An unspecified characteristic that is inherited assumes the value it had at the level of its parent. In the example of Listing 1, the font family of the head is inherited from its parent (the section). If the font family characteristic for section is changed, this will in turn affect the head. This method of determining the value of an unspecified characteristic has to

---

```
\def\section{\starteic{section}
\font{\def\family{cm}}
\presp{\nominal=30pt
\minimum=30pt \maximum=30pt}
\textbrk{\startln=0pt \endln=0pt}
\content}
\def\endsection{\endeic{section}{}}

\def\sectionhead{\starteic{head}
\font{\inherit=1 \def\style{sanserif}
\size=14pt \def\weight{bold}}
\leading{\lead=14pt}
\quadding{\def\quad{right}}
\keeps{\keep=1 \next=1}
\postsp{\nominal=24pt
\minimum=20pt \maximum=30pt}
\textbrk{\startln=0pt \endln=0pt}
\content}
\def\endsectionhead{\endeic{head}{}}

\def\firstrpara{\starteic{para}
\indent{\firstln=0pt}
\textbrk{\startln=0pt \endln=0pt}
\content}
\def\endfirstrpara{\endeic{para}{}}

\def\nonfirstrpara{\starteic{para}
\indent{\firstln=15pt}
\presp{\nominal=6pt
\minimum=4pt \maximum=6pt}
\textbrk{\startln=15pt \endln=0pt}
\content}
\def\endnonfirstrpara{\endeic{para}{}}


```

---

**Listing 4.** Translation of a FOSI Fragment.

be explicitly requested by setting the `inherit` attribute of the affected category to one, as shown in Listing 2. Explicitly assigned characteristic values override inherited values.

The usual method of determining the value of a characteristic that has not been explicitly assigned in the e-i-c is to look up its value in an environment. Every FOSI contains the document environment that explicitly mentions all 128 formatting characteristics. This is the default or “unnamed” environment normally used when a lookup must be done. For example, the prespace category (presp) was entirely omitted from the declaration for head in Listing 2. So head was typeset using the default environment’s prespace characteristic values, which were all zero.

Other “named” environments may optionally be defined in the envdesc section. For an e-i-c’s characteristic to be looked up from a named environment, the structure in an e-i-c that contains the categories (charlist) must set its `envname` attribute to the environment name.

Of the two methods of determining the values of unspecified characteristics (inheriting from a parent and defaulting from an environment), the inheritance method is the more problematic. Since the value of an inherited characteristic cannot be decided until the element’s context is known, current characteristic values must be tracked by TeX. Fortunately, TeX’s grouping already works this way. The characteristic values that must be looked up from an environment can be added to the definitions in the FOSI as part of the translation process, or the lookup can be performed by TeX as part of the typesetting process.

## Typesetting the Translated SGML Document

The processes performed by TeX that culminate in typesetting the translated document can be separated into two levels. The top level is responsible for the inheritance, lookup, and setting of characteristic values, as discussed above. Macros, such as `\starteic` and `\endeic` used in Listing 4, group these values to restrict inheritance, while `\font`, `\textbrk`, and the like are used to set explicit overrides.

The bottom level is responsible for the setting of TeX parameters. This layer is invoked at the end of every start tag. In Listing 4, it is the call to `\eiccontent` that triggers this processing.

Various optimizations are possible. For example, if the only category changed since the last text

fragment is the leading category (which controls line spacing), then there is no reason to change the current font. By keeping track of the categories that have not changed since the last time the bottom layer was called, we save the overhead of computing any TeX parameter that relies entirely on those unchanged categories.

Whatever optimizations are used, it is required that the current font, horizontal and vertical sizes, margins, indent, interword space, page and line breaking, and baselineskip parameters be properly set. Some non-primitive parameters (for example, for controlling the number of columns) must also be set. In addition, certain TeX commands, such as inserts, vertical and horizontal skips, counter increments, macro text expansions for typesetting, and so on, must be executed at the appropriate times. All of these actions must conform to the current settings of the FOSI characteristics.

Sometimes the correspondence between FOSI characteristics and TeX capabilities is close, and a simple transformation will allow TeX to produce the results specified by the FOSI. An example is the transformation of the pre-space category (presp), which controls vertical spacing. Presp contains characteristics, called minimum, nominal, and maximum, that specify the whitespace that precedes an e-i-c. The actions TeX must take can be defined by means of the transformation:

```
<presp nominal=x minimum=y maximum=z> →  
 \vskip x plus min(z-x,0) minus min(x-y,0)
```

The indent category’s characteristics are also easy to transform into TeX. There are only three indent characteristics, all of which are dimensions: `leftind`, `rightind`, and `firstln`. It is possible to specify that a dimension be absolute or relative to its current value. So, assuming that the conditional `\ifabslind` is set to false if the `leftind` is specified relatively and to true if it is specified as an absolute value, and likewise assuming that `\ifabsrind` and `\ifabsfind` are appropriately set, the transformation becomes:

```
<indent leftind=x rightind=y firstind=z> →  
 \ifabslind\else\advance\fi\leftskip x  
 \ifabsrind\else\advance\fi\rightskip y  
 \ifabsfind\else\advance\fi\parindent (z-x)
```

Another fairly straightforward transformation between FOSI characteristics and TeX parameters is the font assignment. The FOSI font category includes characteristics named `style`, `famname`, `size`, `posture`, `weight`, `width`, `allcap`, `smallcap`, and `offset`. A table lookup scheme can be devised that allocates

the fonts found on the user's system based on the classification given by these characteristics. I would exclude allcap and offset from the classification, as these are not really properties of a font.

## Difficult Transformations

The three transformations listed above are among the easiest. The characteristics affecting one *T<sub>E</sub>X* parameter do not necessarily come from a single category. Sometimes the transformation into *T<sub>E</sub>X* requires a long and complex algorithm. The seemingly simple request `<span span=1>` would cause an element to interrupt the current column mode in a multicolumn document, balance off the existing text on the page, switch into one-column mode for the duration of the element contents, and then switch back into the interrupted-column mode. These changes would also affect any *T<sub>E</sub>X* parameter whose setting depends on the `\hsize`. Nonetheless, multicolumn algorithms exist and the required side effects of switching column modes can be rigorously determined. So the span characteristic can, in theory, be implemented.

There are characteristics that are impossible to implement in *T<sub>E</sub>X*: The category that controls page breaks (keeps) contains the characteristics `keep`, `widowct`, and `orphanct`. The first is a toggle (0 or 1) that inhibits the breakability of the entire e-i-c. The other two are integers that control the number of widow or orphan lines to be kept together if the element must break. But *T<sub>E</sub>X* only provides widow/orphan control for page breaks between the first two and the last two lines of a paragraph. So the best transformation is only approximate:

```
<keeps keep=t widowct=a orphanct=b>  -->
\ifcase t \interlinepenalty=10000
\else
  \widowpenalty=\ifnum a>1 10000 \else 150 \fi
  \clubpenalty=\ifnum b>1 10000 \else 150 \fi
\fi
```

The lettersp category concerns kerns between letter pairs. *T<sub>E</sub>X* can be made to do "track kerning" in limited circumstances, but the process is inefficient and the conditions under which it can be used are limited. There seems to be no point in attempting to implement this capability.

The quadding category controls justification of lines within an element. Among other possibilities, it gives the FOSI designer the power to request that paragraph lines be ragged on the inside margin only or the outside margin only. But *T<sub>E</sub>X* cannot justify the lines of a single paragraph based on which page they fall on, at least not in a one-pass system. This

is yet another esoteric request that would not cause a book designer to lose any sleep if it were glossed over.

Still other FOSI capabilities can be implemented by using extensions to *T<sub>E</sub>X*. The category that controls underscoring and overstriking (`highlt`) may require a *T<sub>E</sub>X* extension or some driver assistance via `\special` commands. This same category gives control over the background and foreground colors.

## *T<sub>E</sub>X* Capabilities That Are Not Expressible In a FOSI

It is interesting to note that just as there are FOSI capabilities that are not possible to implement by *T<sub>E</sub>X*, there are *T<sub>E</sub>X* capabilities that cannot be described in a FOSI.

The `plain.tex` package already provides many typographical parameters to which the FOSI designer will have no access. Only parameters and capabilities that may need to be used in the middle of a document will be listed, since the macro package can set up the other parameters easily. The list includes: horizontal kerning; `\vboxes` and `\hboxes` to any fixed dimension; the capabilities of `\halign`, `\valign`, and simple tabbing; mathematics and all parameters related to mathematics; `\looseness`, `\parshape`, and the paragraph-hanging parameters; `\lineskip` and `\lineskiplimit` control; `\topskip`; multilingual hyphenation patterns; marks of various flavors; and `\xspaceskip`, although interword space can be adjusted.

Adding macro packages increases the shortcomings of the FOSI. Add to the list: mixed multi-column modes on one page, although spanning to one column is possible; precise control of figure placement and many insert categories; side-by-side paragraphs; "picture" modes; multiple levels of footnotes; marginal notes; paragraph line numbering. The list goes on.

In general, the major advanced capabilities that *T<sub>E</sub>X* has over FOSI capabilities are macro expandability, conditionals, and the ability to define custom output routines. For the time being, these are not serious limitations. It is more important to find an interim solution to the arbitrary DTD for-matting problem. The FOSI-driven *T<sub>E</sub>X* formatting engine provides a good solution. Its wide acceptance in the SGML community would also mean a wide acceptance of *T<sub>E</sub>X*, a factor that would weigh strongly in *T<sub>E</sub>X*'s favor.

# Specifying Document Structure: Differences in L<sup>A</sup>T<sub>E</sub>X and TEI Markup

C. M. Sperberg-McQueen  
ACH/ACL/ALLC Text Encoding Initiative  
University of Illinois at Chicago  
Internet: U35395@uicvm.cc.uic.edu  
Bitnet: U35395@UICVM

## Abstract

L<sup>A</sup>T<sub>E</sub>X and the Standard Generalized Markup Language (SGML), specifically the SGML tag set created by the Text Encoding Initiative (TEI), are two major systems developed to make it easier to create and verify valid documents. Each attempts to specify and enforce explicit definitions of valid textual structures; each faces questions regarding the structural components of texts, as well as the choice of abstract structures for representing and of formal notations for specifying them.

This paper focuses on the ways L<sup>A</sup>T<sub>E</sub>X and the TEI identify and classify the structural and other components of text; discusses the models of text underlying the two systems and the methods of text definition and validation they make possible; describes a number of specific issues that arise; considers some systematic differences; and describes one possible way in which they might coexist.

## Introduction

As mechanical processing of text becomes easier, it also becomes easier—and more important—to specify formally what a text is and to use that specification to ensure the validity of the data stream that represents the text in the machine. Validation becomes important because application software uses increasingly complex data structures for text representation, and because our mechanical processing can destroy or corrupt data with an efficiency and thoroughness that far exceed the wildest dreams of the most assiduous scholar working by hand. Validation has become easier because computer science has provided a rich set of data structures to use in representing texts and increasingly sophisticated notations for specifying the valid forms of those data structures.

Today I want to discuss the specification of document structure in L<sup>A</sup>T<sub>E</sub>X and in the SGML tag set defined by the ACH/ACL/ALLC Text Encoding Initiative (TEI), an international effort to define an application-independent, language-independent, system-independent markup language for general use (especially in research). This has four parts:

First, I'll discuss the substantive questions of what the structural components of texts are; and, second, the methodological questions of choosing abstract structures with which to represent texts and formal notations with which to specify the abstract structures. Third, I'll describe briefly a number of concrete problems in the proper application of such abstract structures and formal notations to pre-existing texts of the sort studied by most textual scholars, and, finally, I'll describe how I think SGML and L<sup>A</sup>T<sub>E</sub>X can usefully coexist in practice.

Any text-encoding scheme must provide ways to represent the characters of a text, its basic structure, intrinsic features other than structure, and extrinsic information associated with the text by an annotator. I am here concerned not with the first of these, but only with the other three.

## Substantive Issues: What Belongs in a Text?

**Basic text structure.** On the basic structural components of text, there is a rather surprising agreement among the various markup languages in

current use—at least among those which attempt to assign structure to texts.

*LATEX* implicitly divides a text into a title page (created by the `\maketitle` command, which must be preceded by author, document title, and similar information), followed by the text body and, optionally, by back matter (marked with the `\appendix` command). The body and back matter comprise either undivided text or a series of `\parts` or `\chapters`. Within parts, there is a straightforward hierarchy of chapter, section, subsection, subsubsection, paragraph, and subparagraph, in which the hierarchical relationships are enforced automatically.

The TEI tag set similarly divides documents into front matter (which can contain more than the title page), body, and back matter, with body and the parts of the front and back matter all divided into hierarchically nested blocks of text. Since existing (historical) texts may use structural units with names other than *chapter*, etc., TEI uses the generic term *div* for these blocks of text: The text body is a series of `<div0>`s, divided into `<div1>`s, divided into `<div2>`s, etc. The user can specify what name should be associated with a given level by giving the name as the value of an SGML attribute on the tag; for example, `<div1 name='Chapter'>`. The current draft stops at `<div5>`, but this is a purely arbitrary decision and can be changed.

An alternative proposal (used in some existing SGML tag sets) is to eliminate the redundant nesting-level numbers and replace `<div0>` through `<divN>` by the single tag `<div>` or `<block>`. Since the nesting level can be readily calculated at processing time, blocks at different levels can be processed differently. This is elegant but complicates life for whoever is specifying the processing.

**Lower-level floating structures.** Within the main structural divisions of the document, text is divided into paragraphs, and these have no visible internal formal structure. There are some chunks of text, however, that do have visible internal structure; these I call *crystals*, borrowing a term from Steven J. DeRose (in a TEI working paper). Crystals are internally structured free-floating units of text, such as figures, tables, or bibliographic citations. Leslie Lamport calls (some of) them *floating bodies*.

*LATEX* and the TEI recognize roughly the same set of large-scale crystals: lists, verbatim examples, displayed equations, figures, tables, and bibliographic references. The TEI further expects to provide tags for marking much smaller crystal struc-

tures like dates, addresses, personal and corporate names, and so on. This reflects a major difference between *LATEX* and the TEI: *LATEX* does not need special markup for addresses or personal names, because these do not typically require special treatment in document layout. The closest *LATEX* gets are with the conventions used by *BIBTEX* to distinguish first names from last names based on where one puts the comma. The TEI is not exclusively or primarily concerned with producing hard copy from documents, but with making it possible to mark the documents' logical structure in support of *whatever* kind of processing the user might want to do. Historians, librarians, office-automation people, and others may all want special processing based on the internal structure of names and dates—not for printing, perhaps, but for indexing or other reasons.

For the converse reason, the TEI has not yet made any concerted attempt to provide yet another language for the description of mathematical equations, figures, graphics, or tables. *LATEX*, being concerned with processing for output (as well as with the logical structure of the text), can hardly get by without providing markup for such crystals. The TEI has thus far exploited a feature of SGML that allows sections of the text to be marked up in non-SGML notations so they can be processed by some appropriate processor. This keeps SGML out of the graphics-standards wars and allows designers of SGML tag sets to stay out, too. Although tables often have a clear logical structure, and it would make sense to attempt to capture this in descriptive markup, the TEI has yet to make any concrete recommendations in this area; this is an area of ongoing work.

For bibliographic citations, the TEI provides a structured form patterned on the standard forms for bibliographic references developed by librarians, as well as a much less tightly structured form for those with less concern about database usage of their citations. The structured form provides more structure than appears to be available in the prose segments of *LATEX* documents, but is less rich than the corresponding *BIBTEX* structure. This is an area in which the TEI tags must definitely be extended to at least the level of detail offered by *BIBTEX*.

**Phrase-level attributes.** Within the paragraph, the rigid hierarchical text structure of chapter, section, subsection, etc., suddenly breaks down, and we are confronted with a non-rigid mess with the consistency of soup. Within this soup, some larger chunks (crystals, like figures and tables) may be

floating that we've already discussed. Some non-structured bits may be floating there as well: emphasized phrases, quotations, and the like. Here, L<sup>A</sup>T<sub>E</sub>X and the TEI take a very similar approach. Instead of describing the visual presentation of the text in a particular output medium, both encourage the user to describe its logical characteristics. Thus, L<sup>A</sup>T<sub>E</sub>X provides an `\em` command for emphasized text and suggests that the `\bf`, `\sc`, and similar commands "should appear not in the text but in the definitions of the commands that describe the logical structure." Similarly, the TEI provides several tags for marking words, phrases, or passages that are specially marked in some way:

- `emph`
- `foreign`
- `cited word`
- `term`
- `book or journal title`
- `quotation`
- 'scare quotes'
- `article or poem title`

In addition, since for historical texts one doesn't always know why something is presented in a different font, one can also mark such material simply as `<highlighted>` without any attempt to explain why. This is a necessary compromise between the advantages of logical or descriptive markup and the requirements of scholarly integrity.

**Typographic details, layout, processing.** Treatment of typographic details, layout, and similar matters is predictably far more elaborate in L<sup>A</sup>T<sub>E</sub>X than in the TEI tag set. L<sup>A</sup>T<sub>E</sub>X, even with its explicit preference for *logical* document design over *visual* design, does after all have the function of providing good typeset output; since good typesetting is not wholly algorithmic, T<sub>E</sub>X and L<sup>A</sup>T<sub>E</sub>X provide plenty of opportunities for the user to give them hints on what the output should look like.

The TEI tag set is far poorer in this respect, for two reasons: First, we are attempting to create an application-independent markup scheme, suitable for many different types of processing. It seems more important just now to stress the possibilities for processing other than printing, because these are so often overlooked. Trying to provide a rich set of layout tags in the first draft would invite serious misunderstandings and suggest that the TEI was trying to compete with T<sub>E</sub>X and other typesetting systems. The second reason is that SGML is designed as a declarative, not a procedural language—precisely to ensure the application independence it is

designed to achieve. It *is* possible to specify presentation declaratively rather than procedurally, as we do already with the `<highlighted>` tag described above. But a full declarative description of page layout is a large, challenging assignment, one that requires a lot of further work. It is also a task that the International Organization for Standardization (ISO) is already addressing with its Document Style and Semantics Specification Language (DSSSL); if the DSSSL project is successful, the TEI can piggy-back on their success by basing its further work on layout problems on DSSSL.

**Annotation.** L<sup>A</sup>T<sub>E</sub>X provides useful tools for annotation: footnotes, marginal notes, and (in SLiT<sub>E</sub>X) inline display notes. These correspond directly to a single TEI tag, `<note>`, that uses an attribute value to specify its location or type. But the TEI provides a large number of other tags for annotation of various kinds that do not appear in L<sup>A</sup>T<sub>E</sub>X:

- an extensive document header that documents the electronic text: its date and place of origin, names of those responsible, copy text used, and specifics of the encoding used;
- tags for special items, like dates and numbers, that allow their values to be given in a standard format (so that a note containing the sentence, "Let's have lunch next Thursday," might tag "next Thursday" as a date with the standard value 18 July 1991 or, in ISO format, 1990-07-18);
- tags for recording editorial interventions, such as corrections in the text, normalized spelling, additions, deletions;
- page and line references to canonical editions;
- text-critical apparatus; and
- most notably, a set of tags for the specification of linguistic analysis or other interpretive material relating to a text, which can be used (for example) to specify part of speech information or syntactic structure for every word or sentence of a corpus.

This wealth of annotation markup reflects, of course, the particular interest in analysis and interpretation of existing texts found in the research community, the need for which led to the creation of the TEI as a project.

In all, L<sup>A</sup>T<sub>E</sub>X and the TEI tag set present a fundamentally similar view of the major components of text; they have much the same view of basic text structure and provide similar facilities for handling most of the phrase-level markup needed for prose. They differ in the amount of attention paid to figures, tables, and similar matter; in the amount of

detail possible for the typographic description of the text; and in the richness of their facilities for annotation. These differences reflect in part the difference between those interested in technical documentation on the one hand (which I take to be the original audience of *LATEX*) and those interested in the study and analysis of existing texts, which is the constituency of the TEI. In part, they reflect the difference between a mature piece of software aimed at a particular kind of processing, and a markup scheme still in progress designed to be independent of any particular application and any particular piece of software; and in part, these differences reflect a slightly different model of what text is. It is to this difference that I now turn.

## Models of Text and Text Grammars

Any markup language must embody some idea of what text is, formally. How complex and how suitable that idea is for formal processing vary, of course.

Some languages (especially early ones) equate *text* with internally unstructured strings of characters; often this unstructured character string is punctuated by occasional processing instructions that themselves are constrained only by specific implementation details. When no processing instructions are allowed, you have *ASCII-only text*, in which markup is limited to the command repertoire of a 1956 Teletype machine (carriage return, vertical and horizontal tab, backspace, and bell).

For serious processing, extensive command sets have been developed, mostly oriented to the task of getting ink on paper in the right places. Commonly known schemes of this type include Waterloo and IBM Script, troff, most word processors, and, of course, *TEX*. Processors built on this model are flexible and very easy to understand, but very difficult to prove correct. The number of states in which such a processor can be explodes with the number of commands, and there can be very tricky interactions among various states. Since the state of the system at any point is a function of the entire document up to that point, it is hard to process documents in languages like this except by left-to-right scanning. And since almost any string of characters and commands is legal, these languages offer no real help in verifying the structural validity of machine-readable documents.

A dramatic reduction in the combinatorial explosion of possible states comes with systems that view text as a block-structured hierarchy. The hierarchy is typically a relatively simple one of two or three levels, below which one is back in a sort

of primordial prose soup without visible structure. Well-known markup languages in this class include IBM and Waterloo GML, various macro languages for Script and troff, some style packages for micro-computer word processors, and, of course, *LATEX*.

These languages introduce a new (hierarchical) model of text, and can thus avoid some interactions among states by simply declaring them illegal. Thus, in *LATEX*, it is not legal to have a document body without an enclosing document environment, and, in Waterloo GML, the software checks to ensure that the front matter does not follow the back matter. But no formalisms are introduced to make the document hierarchy fully explicit; there is no explicit *document grammar*. It is naturally impossible then to enforce document validity fully or automatically. Waterloo GML does not check to see that the back matter does not precede the body of the document. Since the more rigid notion of valid document structures is not consistently enforced, these document languages are a bit like programming languages with weak type systems and automatic type coercion and control structures built around the *GOTO*, by relying on the user to follow *good practice* rather than by verifying that good practice formally and mechanically. The constraints which are enforced are hard-coded into the processing code and can thus be hard to change.

The next distinct model of text visible in text processing uses fully explicit, well-defined hierarchies of text elements to define legal text structures. In some cases, like Word Cruncher markup, the hierarchy is so simple that there may still be no explicit specification of the underlying document grammar; in others, the legal structures of documents are specified explicitly and can thus be enforced formally. The best-known markup scheme in this class is the Standard Generalized Markup Language (SGML), which differs from its prototype (IBM GML) precisely in the addition of explicit document grammars with context-free power. (Strictly speaking, of course, SGML is not a markup language but a meta-language that allows the definition of markup languages, precisely because it provides an explicit language for the expression of document grammars.)

SGML markup is of two types: Structural units of the text or specific points in the text (*elements* in SGML parlance) are indicated with SGML *tags*, delimited conventionally by angle brackets or by left-angle-bracket-plus-slash and right-angle bracket. Segments of the text are delimited by a *start-tag* and an *end-tag*, much the same way structural units in *LATEX* are delimited by `\begin{environment}` and

\end{environment} commands or by left and right braces.

The second type of SGML markup, *entity references*, allows one to insert characters in a document by referring to an *entity* that contains those characters. Entity references can thus be used for special characters not on one's keyboard (analogous to L<sup>A</sup>T<sub>E</sub>X's commands for accented letters, etc.), for include boilerplate language (analogous to user-defined macros in T<sub>E</sub>X that insert formulaic language into a document), and to include external files (analogous to L<sup>A</sup>T<sub>E</sub>X's \input and \include commands).

Any markup used in an SGML document must be explicitly *declared* in a *document type declaration*. Entities are declared by specifying their name and the replacement value (which can be the name of a system file or a string of characters). Elements are declared by specifying their name and their allowable *content*; the declaration for element X specifies what can occur within an X (or within the scope of an X tag), such as character data, other tags, etc. The document type declaration is thus similar to a grammar that specifies the legal forms of a document of a given type; the individual declarations correspond to the production rules of a grammar in Backus-Naur Form (BNF).

The SGML element declaration, however, uses a slightly richer notation than BNF. The *content model* of an element is (more or less) a regular expression composed of the names of SGML elements and the keyword #PCDATA (*parsed character data*). SGML thus resembles a regular right-part grammar more than BNF does, but there are further wrinkles we need not go into here that can make SGML content models slightly more compact than regular right-part grammars.

The use of an explicit grammar, together with the explicit delimiters for enclosing each SGML element, leads to a natural view of an SGML document as a tree rather than as a simple unstructured string. The complexity of the processing is contained, since the grammar is basically context-free, and the state of the system at any point in the text can be read by traversing the tree from the root node. L<sup>A</sup>T<sub>E</sub>X documents (like any documents with a block-structured model of text) can be treated this way, but the absence of any explicit grammar tends to make such treatment a purely academic exercise.

## Specific Design Issues

Some design issues arise in any attempt to specify a document structure that is at once rich and flexible

enough to be usable in practice and rigid and precise enough to be formally verifiable.

**Prescription and description.** First of all, one encounters a fundamental tension between prescriptive and descriptive specifications of document structure. If one is purely prescriptive, one can ensure that the documents one processes will all have very similar structures. Software can make good use of this consistency. However, when one is encoding an already existing text written by someone else—possibly long dead—it is fruitless to expect it to match a specific prescriptive document style, and historically misleading to try. Rigid formal document specifications may fail to match the chaotic reality of historical documents; unless we are willing to violate the historical integrity of the texts we are studying, we have to provide a more flexible formal structure within which we can find a representation even for unconventionally structured texts.

Excessive flexibility means, of course, that the document grammar may allow spurious document structures that never would occur in practice. Given the choice between excessive rigidity, which makes some documents unrepresentable unless the grammar is loosened, and excessive flexibility, which makes some errors undetectable unless the grammar is tightened, the TEI has consistently chosen excessive flexibility. The issue does not arise in this form for L<sup>A</sup>T<sub>E</sub>X, because it does not claim to provide a markup language for arbitrary existing texts; it is comfortable, therefore, with its current degree of prescriptiveness.

**Controlling complexity through modularity.** Whenever a document grammar is rich enough to handle real texts with serious markup problems, it has enough markup primitives to begin confusing users and developers. It is useful, in this case, to group tags into tag subsets that can be defined and understood independently of each other; this helps control the overall complexity of the markup scheme. Of course, it helps a lot if the software can guarantee that tags in different subsets don't have long-distance interactions. We can see such modularity in L<sup>A</sup>T<sub>E</sub>X in the separation of the specialized tags needed for slides and bibliographies into the semi-detached units of SL<sup>A</sup>T<sub>E</sub>X and BIB<sup>A</sup>T<sub>E</sub>X. In the TEI, similarly, the tags for specialized uses are entirely separate and have no interaction with the core tags for phrases and the like. Linguistic analysis, text criticism, editorial intervention, etc., can all be turned on or off by the user. The current direction of development will lead to more such encapsulations in the next version of the TEI DTDs.

The user, of course, may need to use arbitrary combinations of these specialized tag subsets together; this requires a careful specification of their semantics to avoid side effects.

**Multiple hierarchies.** Although most texts fall comfortably into a hierarchical analysis of their parts, the use of cleanly hierarchical, block-structured markup does lead to problems whenever the text falls comfortably into more than one such hierarchical structure. The volume, page, column, and typographic line numbers of a standard edition form a simple, clean hierarchy, but one which probably does not nest well with the logical hierarchy of part, chapter, section, paragraph, sentence, and word. If there are several standard editions whose page references should be noted, we have one hierarchy for each edition. When the text is in verse, we can add the metrical hierarchy of canto, stanza, line, and foot. And, of course, the labors of scholars may assign rhetorical, thematic, narrative, and other structures to the text.

The TEI scheme uses the SGML feature of *concurrent markup* to allow the user to maintain several hierarchies in the same document. Bound by the strict block structuring of *TeX*, it is hard to see any solution to this problem for users of *LATEX* except to choose one hierarchy as the main one, and to reduce the other hierarchies to simple scope-less declarations in the text.

**Systemic comparison of SGML and LATEX.** *LATEX* and SGML resemble each other strongly in their common goals of providing system- and device-independent markup and processing for texts, and in their basically similar hierarchical models of text. SGML pushes the hierarchical model and the notion of formally specified, verifiable document structure farther than does *LATEX*. It provides a mechanism for formal specification of a document grammar, and validates the document automatically against that grammar.

SGML attempts to provide a notation that is not only system- and device-independent but also software- and application-independent. The origins of SGML are in attempts to ensure the reusability of machine-readable texts by divorcing markup from processing, and stressing descriptive or logical markup rather than procedural markup. *LATEX* stresses the utility of logical markup to ensure the structural soundness of a document and to make it easy to lay it out in different styles. SGML and the TEI push that concept farther and stress the importance of logical markup in ensuring that a document can be processed without change for entirely dif-

ferent applications, including applications that have nothing to do with text layout or typesetting.

This insistence on application-independence leads SGML into what is its most striking feature as a markup language: its complete lack of semantics. SGML markup languages are entirely declarative, not least because SGML simply provides no formal mechanisms for defining any non-declarative meaning for them. SGML allows you to say that a given stretch of your document is (say) a quotation. It does not require that you say how you want it processed; indeed, it makes it impossible for you to do so in SGML. You specify how an application program should process an SGML document by talking to the application program, not by talking to SGML. The document itself remains a logical object untouched by specific processing instructions. (N.B.: Inserting processing instructions directly into SGML documents is allowed, provided the instructions are explicitly marked as processing instructions so they can be skipped by other software.)

## Coexistence

The TEI is intended to be an application-independent markup language for texts of any period, any genre, and any language. Because many of its users will need or want to use already existing software for processing their texts, without modifying that software to read SGML documents, the TEI is intended from the outset to coexist with other software-dependent file formats. The fundamental similarities of goal and the basic harmony of their common emphasis on the logical structure of text combine to make it very simple for the TEI scheme to coexist with *LATEX* in a single system.

Any file stored locally is stored in some particular file format. This *local storage format* may or may not be identical with the input format of any application program. If only one application is run on it, the file is almost certain to be stored in that application software's input format. A document processed repeatedly with several different packages, however, might have its own format, from which it is translated into the input formats required by the software.

One obvious use for a scheme like the TEI tag set is as a local document storage format. When one wants to make a concordance from a document, one translates it from the TEI format into the form required by the Oxford Concordance Program or some other concordance package; when one wants to make

hard-copy output, one translates it into the form required by the desired formatting or typesetting program. The structural similarities of the TEI scheme and L<sup>A</sup>T<sub>E</sub>X mean that a TEI-to-L<sup>A</sup>T<sub>E</sub>X conversion is relatively straightforward, and for the most part the same may be said of a L<sup>A</sup>T<sub>E</sub>X-to-TEI translation.<sup>1</sup> In other words, L<sup>A</sup>T<sub>E</sub>X is a natural choice for the typesetting of TEI-tagged documents, just as the TEI format is a natural choice for the encoding of a text's logical structure so that it can be processed by many different pieces of software.

### Acknowledgments

The TEI, an international cooperative effort to develop and disseminate a common format for the encoding and interchange of machine-readable texts, is sponsored by the Association for Computers and the Humanities, the Association for Computational Linguistics, and the Association for Literary and Linguistic Computing.

It is funded in part by the U.S. National Endowment for the Humanities, an independent federal agency; DG XIII of the Commission of the European Communities; and the Andrew W. Mellon Foundation.

The work is done by many generous individuals from the community who volunteer their time to serve on the working committees and work groups of the project.

### References

- [1] DeRose, Steven J. "Suggestions for improving the AAP tag set." TEI working paper, document TEI TR R7, August 1989.
- [2] Knuth, Donald. *The T<sub>E</sub>Xbook*. Reading, Mass.: Addison-Wesley, 1984.
- [3] Lamport, Leslie. *L<sup>A</sup>T<sub>E</sub>X: A Document Preparation System*. Reading, Mass.: Addison-Wesley, 1986.
- [4] Sperberg-McQueen, C. M., and Lou Burnard, eds. *Guidelines for the Encoding and Interchange of Machine-Readable Texts*. Text Encoding Initiative, Chicago, Oxford, draft version 1.1 edition, 1990.

---

<sup>1</sup> This document, for example, was drafted using SGML tags and converted to L<sup>A</sup>T<sub>E</sub>X for submission.

# A Structured Document Preparation System—*AutoLayouter*

## Version 2.0—An Enhancement for Handling Multiple Document Types

Takashi Kakiuchi, Yuki Kusumi, Yoshiyuki Miyabe, and Kazu Tsuga

Information and Communications Research Center

Matsushita Electric Industrial Co., Ltd

1006 Kadoma, Kadoma-shi, Osaka 571 Japan

+81 6 906 4873; FAX: +81 6 906 8148

Internet: [kakiuchi@isl.mei.co.jp](mailto:kakiuchi@isl.mei.co.jp)

### Abstract

*AutoLayouter* is a structured document preparation system used to increase efficiency in creating and reusing designed documents in offices. *AutoLayouter* consists of an easy-to-use structured editor and a Japanese LATEX-based formatter. With a structured editor, the user need not be concerned with page layout, and can concentrate on creating the contents of the document. Because these documents are structured logically, they can be easily reused or processed further by other systems.

At the 1990 TUG meeting, we presented *AutoLayouter* version 1.0. Since then we have been improving the system to handle more complicated document structures, such as are defined in SGML. In this paper, we describe 1) new document structures, and 2) ALTEX, which directly formats structured documents.

### Introduction

Recent research projects on document processing have been directed at structured document representations, such as SGML. The basic idea of a structured document is to separate a document into structure and content; its contents are then extracted in terms of its structure. In an SGML document, the structure is defined explicitly as a DTD (Document Type Definition), so that documents created with the same DTD are interchangable. Such a structure can also be used by a document processing system to retrieve the required information: for instance, the title, author, and date of technical reports can be retrieved through their structure and merged into a summary table.

The structured document representation, especially the logically structured one, is essential to making the best use of electronic documents. We can store documents in electronic format, and load and print them on paper, using conventional word processor or desktop publishing systems. These documents cannot be processed by other systems, however, unless the logical meanings of their contents are preserved, because there is no other way to identify the contents. Because of its abstract, declarative language, LATEX is often referred to as an example of

a text formatter for logically structured documents. LATEX is used as a document preparation tool by computer software engineers because they can use any editor and can concentrate on a document's content and structure without paying any attention to its physical appearance.

In Japan, the advance of word processing technology has meant that business documents are prepared and stored electronically, but they must also be kept in printed form. The format of most Japanese business documents separates items with rule lines. This standardizes the items to be written and determines the text area available for each item. Japanese word processors possess some characteristics for editing these forms: they draw ruled lines and insert text in the area surrounded by the rules. However, this augmentation of rule-line functions has made it too complex to manage document files and to reuse document contents. As a result, a document must still be managed in the printed form, even though it is stored in an electronic format.

To solve these problems, we have developed a structured document preparation system, *AutoLayouter*, whose objective is to increase efficiency in creating and reusing preformed documents. *AutoLayouter* consists of a structured editor for creating SGML-like documents, and a Japanese LATEX-

based formatter called ALTEX. In the subsequent sections of this paper, we mainly describe the document structures of *AutoLayouter* and implementation issues of ALTEX formatter.

## Document Structure

**Model for document structures.** The *AutoLayouter* document is represented as a tree structure (like an SGML document). Each node of the document tree, except the leaves, has a unique label associated with it. Each leaf of the document tree contains a text segment, which is represented as a sequence of characters. Any node may have an arbitrary number of attributes, represented as name-value pairs.

A major difference between the document structure of *AutoLayouter* and SGML is that the *AutoLayouter* document has two structure layers, namely the *logical structure* and the *generic structure*. The logical structure presents the logical meaning of the subsidiary structures, such as a sender's address in a letter, which is specific to the document type. Meanwhile, the generic structure presents such document elements as itemization, enumeration, and centering; these are common to all document types. The generic structure is already predefined in the system. When defining a document structure, we need only specify the logical structure.

The whole document structure is organized as follows: the root node of the document belongs to the logical structure, and its descendants can belong to either the logical structure or the generic structure, according to the document definition. A node in the generic structure cannot be a parent of any nodes in the logical structure; furthermore, siblings belong to the same structure. In the rest of the paper, we shall call nodes in the logical structure the *logical element*, and nodes in the generic structure the *generic element*. Each leaf of the document is a special generic element that has only a text segment with no children.

A model for structured documents should be well designed so as to make it easy to define document structures and maintain consistencies between them, and also to make its editor easy to use. In SGML, the whole document structure must be defined explicitly, using the fully expressive description language. This means that to use the contents of one document in another document, the structure definitions of both must be strictly consistent with each other; such consistency requires as much effort as does designing database schemes. Furthermore,

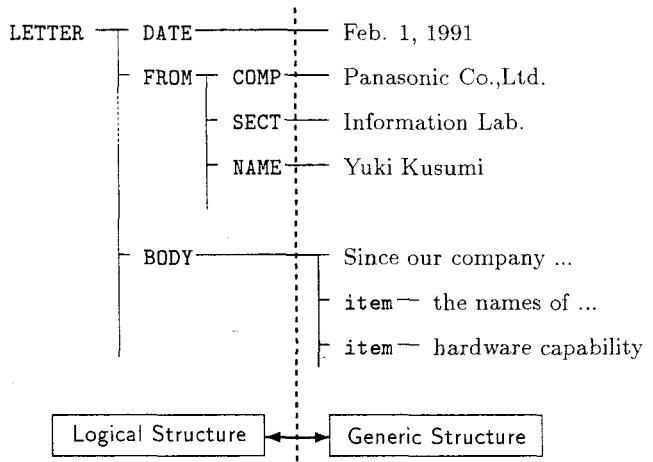


Figure 1: Document structure in *AutoLayouter*

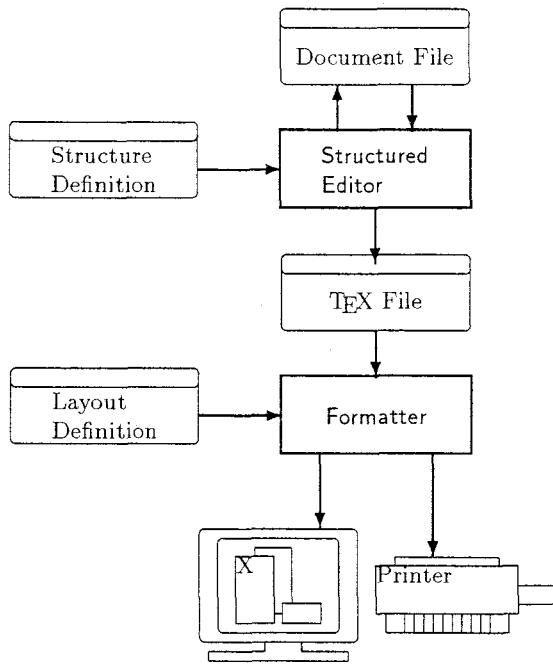
the user interface of a structured editor tends to be awkward because of the flexibility required to handle all document structures as generated from their definition. This is analogous to the trade-off between functionality and ease of use involved with most systems, namely, easy-to-use tools can be achieved at the expense of their restricted flexibility.

In *AutoLayouter*, the generic structure is predefined in the system and only the logical structure needs to be defined; thus, only the logical part of document structures should be designed to be consistent. Moreover, we can build in the easy-to-use, dedicated user interface for editing the generic structure; this contributes to efficiency in preparing documents. A user often manipulates a document's generic structure rather than its logical structure, because most of the logical structure can be generated automatically by the system and need not be modified so frequently, whereas the generic structure contains the text segments to be typed and the layout directives that have been left to the user.

**Example 1:** In Fig. 1, a whole document structure is divided into two structures. The document definition specifies only the logical structure, shown on the left side.

By using these two-layered structures, the design of a new document type is accomplished by defining a logical part of its structure and specifying how to present each element on paper (layout definition).

**Structure definition.** The structure definition of a document type is a generic specification of its logical structures. This is expressed in a grammar format that specifies the logical elements and the order



**Figure 2:** System diagram of *AutoLayouter*

in which they will be generated. Each rule consists of a left-hand side, which corresponds to a node, and a right-hand side, which is a restricted regular expression that specifies occurrences of its children.

### System Structure of *AutoLayouter*

**System overview.** As shown in Fig. 2, *AutoLayouter* consists of two subsystems: a structured editor and a formatter.

The structured editor interactively performs the following tasks:

- interprets a structure definition;
- edits documents, showing the structure elements to be inserted and checking illegal structure modifications;
- loads and saves structured document files; and
- converts documents into *TeX* files.

Meanwhile, the formatter completes the following tasks:

- typesets the document in accordance with the layout definition (style file) provided; and
- converts formatted documents (*dvi* file) to a specified device such as a bitmap display or a PostScript printer.

In the rest of this section, we describe various file formats used by subsystems, to clarify their roles.

**File formats.** The data files used in the *AutoLayouter* are the following:

- a structure definition file (for input);
- a structured document file (for input and output);
- a *TeX* file (for output).
- a layout definition file (for input); and

A *structure definition file*. In order to define document structures (see the Model for Document Structures subsection on previous page), we use the following three syntaxes in the structure definition file.

1. A node having children of logical elements is defined using the following syntax:

```
<!node node_name,
      regular_expression>
```

This implies that if a node is a logical element, then its siblings are also logical elements.

2. A node having children of generic elements is defined using the following syntax:

```
<!leaf node_name, type>
```

A *type* field, which can be *general*, *string*, or *integer*, and so on, specifies a selection of the subsidiary structures that are allowed to appear; *general* allows any kind of generic elements, including any nested sub-tree of a generic structure; *string* allows only a string in a text segment; and *integer* allows only an integer in a text segment.

3. Attributes associated with a node are defined using the following syntax:

```
<!attribute node_name,
           {attr_type
            attr_name = initial_value}*>
```

An attribute, which may be used for any purpose, is typically used to define layout parameters, such as paper size or column layout.

In addition to the syntax above, we provide a syntax just for the structured editor; this is used to define help information for each logical element, such as a label string shown in the editor.

**Example 2:** The following is the structure definition of the document shown in Fig. 1.

```
<!rootnode LETTER, DATE.FROM.BODY...>
<!leaf DATE, date>
<!node FROM, COMP.SECT.NAME...>
<!leaf COMP, string>
...
<!leaf BODY, general>
```

The structured editor reads the structure definition file in two situations: when selecting a document style to create a new document, or when starting to edit an already existing document.

A structured document file. We directly represent a tree structure of an AutoLayouter document as a block structure of the document file. A node  $n$ , whose children are  $m_1, m_2, \dots, m_k$ , is expressed in the document file as follows:

```
<n attribute_list>
  <m1 attribute_list>
  ...
  </m1>
  ...
  <mk attribute_list>
  ...
  </mk>
</n>
```

A *TEX* file. The structured editor outputs a *TEX* file to be input by the formatter. The *TEX* file represents the tree structure of the *AutoLayouter* document directly, converting a node  $<n>, \dots, </n>$  in the document file into a *TEX* command  $\backslash begin{node}\{n\}, \dots, \backslash end{node}\{n\}$ , and replacing all special *TEX* characters with *TEX* commands that generate the characters literally.

```
\begin{node}\{n\}[attribute_list]\{
  \begin{node}\{m1\}[attribute_list]\{
    ...
  \end{node}\{m1\} ...
  \begin{node}\{mk\}[attribute_list]\{
    ...
  \end{node}\{mk\}
\end{node}\{n\}
```

The name of the root node that appears at the top of the *TEX* file identifies the style file.

A layout definition file. The layout definition file is a *TEX* style file. This will be discussed later.

## Editing the Structured Document

As shown in Fig. 3, the editing field of the structured editor is divided into two areas, a style field and a layout field, that represent the logical structure and generic structure, respectively. Usually we use different labels in different structures, such as text labels in *style field* and graphical labels in *layout field*. This makes it easy for users to see the whole document structure. In each field, we use indentations to show substructures.

When creating a new document, one selects the document type, such as *letter* or *report*. The editor reads the structure definition file of the specified document type and generates a mandatory and minimum structure according to the definition rules.<sup>1</sup> Since the mandatory structure has already been gen-

<sup>1</sup> Each leaf of the logical element has a generic element for a text segment.

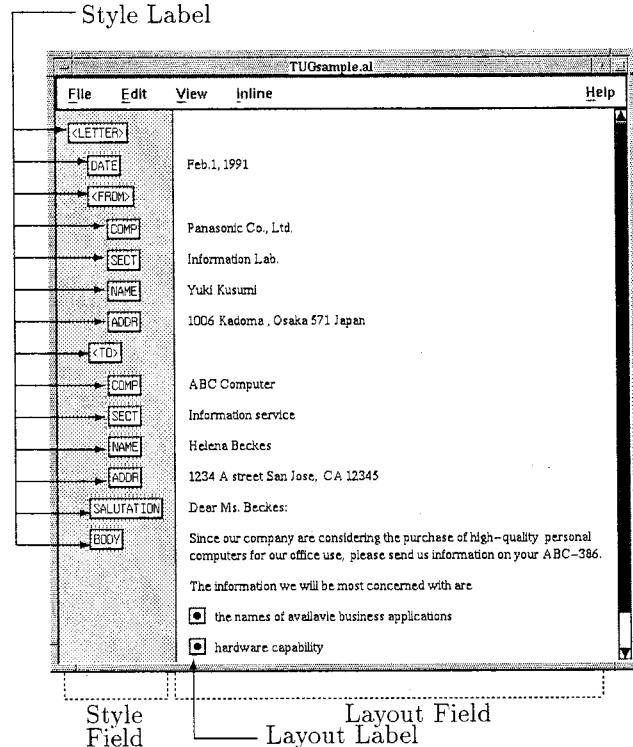


Figure 3: Snap shot of editor screen

erated, one completes the document by simply typing text into each text segment.

One may insert a logical element, such as a report date field, as needed, whenever it has been defined as optional or is repeated in a regular expression. When the insert command is selected for the layout field, the editor displays candidates for the logical elements that can be inserted at the specified position. One only needs to select a candidate to insert it. Since only valid candidates are shown, an illegal structure can never be generated. When deleting a node, the editor checks whether this violates a rule; if it does, the editor displays an error message and ignores the user's operation.

In the layout field, one can insert any generic element at any position, as long as the type of its ancestral logical element is declared as *general* in the definition. When the insert command on the layout field is selected, the editor shows a label list containing all generic elements.

The editor also has additional features listed below:

*Motif as Graphical User Interface.* Motif provides a consistent look and feel in different applications.

*Japanese Kana-to-Kanji conversion.* We developed Japanese input as a front-end processor. Communication between this and the text editor realizes in-line conversion of Japanese.

*Operations with keys.* Most commands can be operated with either a mouse or a keyboard. This satisfies a wide range of users, from novice to expert.

*Browsing functions.* Moving around labels that have a keyboard focus switches the contents of the panels that display the attributes and the help messages.

## Formatting with ALTEX

*AutoLayouter* formats its structured documents using an original typesetter called “ALTEX”, which has the following features:

- handles a tree-structured document directly; and
- provides ready-to-use macros to support layout abstraction.

ALTEX is implemented in LATEX.<sup>2</sup> Therefore, not only can LATEX users include their LATEX documents within an ALTEX document, but LATEX experts can easily describe a layout definition by using LATEX commands.

We will describe our ALTEX in detail with respect to these features in this section.

**Formatting tree-structured documents.** First, we will explain the mechanism for mapping a structure to its layout. As we mentioned in the section System Structure of *AutoLayouter*, a structure element in a document is represented in the form

```
\beginnnode{...},...,endnode{...}
```

in an ALTEX file produced by the structured editor. ALTEX expands the two control sequences \beginnnode and \endnode in the same way that it is used in the LATEX environment, namely \begin{...},...,end{...}. For instance, a structure

```
\beginnnode{foo}[attribute list]{  
...  
}endnode{foo}
```

is expanded to the following:

```
\begingroup\nodefoo{attribute list}{  
...  
}\endnodefoo\endgroup
```

This expansion indicates that the layout for a structure *foo* is based on the definition of two control sequences, \nodefoo and \endnodefoo.

In this mechanism, it should be noted that the text segment of a structure is enclosed with the

---

<sup>2</sup> Japanese LATEX (ASCII version), to be exact.

grouping symbols { and }. The braces allow the text segment to be processed as an argument to a T<sub>E</sub>X macro in some cases, or to be laid out as text as soon as it appears in other cases. To be more specific, in the case where the text segment is to be placed directly into the main vertical list, one can define the control sequence \nodefoo as

```
\def\nodefoo#1{...}
```

In this case, \nodefoo works as a pre-processor before the text segment is laid out on the page. If, on the other hand, the text segment needs processing, or it should be saved once and laid out later, one defines \nodefoo as

```
\def\nodefoo#1#2{...}
```

This form of definition enables us to describe any operations on the text segment (i.e., argument #2) in the replacement text of the macro definition. However, note that the former form is recommended wherever possible, because the latter form consumes more T<sub>E</sub>X memory.

**Example 3:** Let us consider a definition for a declaration of the author of an article, similar to the \author command in LATEX. In the LATEX article.sty file, the \author command is defined as:

```
\def\author#1{\def\@author{#1}}
```

i.e., the \author command saves its argument into a macro \@author. In order to implement the same function as the \author command in ALTEX, we define a \def\nodeAUTHOR macro for a logical structure AUTHOR as:

```
\def\nodeAUTHOR#1#2{\gdef\@AUTHOR{#2}}
```

The mechanism mentioned above is not applied to the outermost structure, namely \beginnnode{root} and \endnode{root}, which represents the root node of the document, because it requires extra tasks. The \beginnnode{root} command should load a layout definition file and set up miscellaneous parameters, and the \endnode{root} command should flush out the main vertical list and process cross-references.

Incidentally, ALTEX expands attribute lists in a uniform fashion. For instance, if an attribute list of the structure *foo* appears as:

```
\beginnnode{foo}[attra=vala;  
attrb=valb]{
```

then each “attribute=value” pair is expanded into a command \foo@attribute{value}, i.e.:

```
\foo@attra{vala}  
\foo@attrb{valb}
```

To process the expanded attribute list, we must prepare control sequences that have one argument “`\foo@attribute`” for each attribute associated with a node `foo` in a layout definition file.

**Layout model and layout definition.** When considering a practical usage for a document preparation system that is based on a structured document, providing a toolkit to facilitate layout definitions is indispensable. When using a document preparation system with WYSIWYG and direct manipulation features, we can perform any page layouts with some cumbersome efforts. Obviously, *AutoLayouter*’s automatic layout feature does not work without a layout definition. This becomes the most critical bottleneck in practical use.

To keep the toolkit from being complex and confusing, it should be based on a well-designed and simple layout model. In ALTEX, we provided two layout models, a *paragraph layout model* and a *form layout model*. Each tool is an abstraction of a layout based on these models.

In the rest of this section, we present these two layout models, as well as the way to use the toolkit to map the logical structure element to the physical layout.

**Layout model.** The sequence of words in a text segment is broken into lines with the paragraph layout model. The result of paragraph layout is a *box* that might either be put into the main vertical list directly, or aligned vertically or horizontally together with other boxes before being put into the main vertical list. In the latter case, the alignment is performed on the form layout model. Now, let us see each model in detail.

*Paragraph layout model.* This model is provided for the sake of putting the contents of a structure element into the heap of lines. Each text segment in the leaf elements contains logical paragraphs. These are put into the physical layout of the paragraphs, whose shapes vary according to the parameters shown in Fig. 4. We utilized *TeX*’s line-breaking mechanism in implementing this model; itemizing, centering, and flushing, for example, can be represented with this model.

Roughly speaking, this model corresponds to *LATEX*’s *list* environment with only one `\item`. However, our model has such extended features that we can set labels on top of the second and subsequent paragraphs, as well as the first one, and we can set the arbitrary shape of any hanging indent, and so on.

Furthermore, when both a node and its children are laid out with this model, the margin of the

parent node is inherited by the children. This is why the layout of nested items is guaranteed, as is expected.

Incidentally, we furnished ALTEX with a command to define a structure as this model. Assume structure `foo` is defined as a node laid out with this model, then the result of `\begin{node}{foo}` ..., `\end{node}{foo}` is put into a `\vbox`, such as the main vertical list, after the text segment in the structure has been broken up into lines.

*Form layout model.* This model is provided to make forms in which boxes are aligned with each other. In this model, the alignment of boxes is modeled as the tree structure shown in Fig. 5(a). Each node of the tree aligns its children either horizontally or vertically. As our approach is based on *TeX*, this model is implemented as nested `\vboxes` and `\hboxes`.

ALTEX also provides commands for making various boxes, as well commands to align the boxes. For example,

- a command to make a box with specified width and height: the layout of the inside of the box can be also specified, along with centering, flushing, paragraph shape, and so on. (See Fig. 5(b).)
- the commands to make a box for the title and to specify the contents for it: the same layout commands have the same function as above. (See Fig. 5(c).)

In plain *TeX*, it is not easy to make a box with a specified width and height, which is why we decided to provide these commands at the system level.

In addition, we created some commands, used instead of `\vbox` and `\hbox`, to improve the readability of the layout definition. Using *AutoLayouter*, one can describe a vertical box with

```
\BeginV
  ...
\EndV
instead of with
\vbox{\offinterlineskip
  ...
}
```

**Two ways to map a structure to its layout.** There are two ways of mapping a logical structure element to its physical layout, namely *direct mapping* and *indirect mapping*, depending on how the occurrence of the element corresponds to its layout.

*Direct mapping.* In the case of the *letter* or *article* style, most of the logical elements are laid out in the same order as they appear in a document.

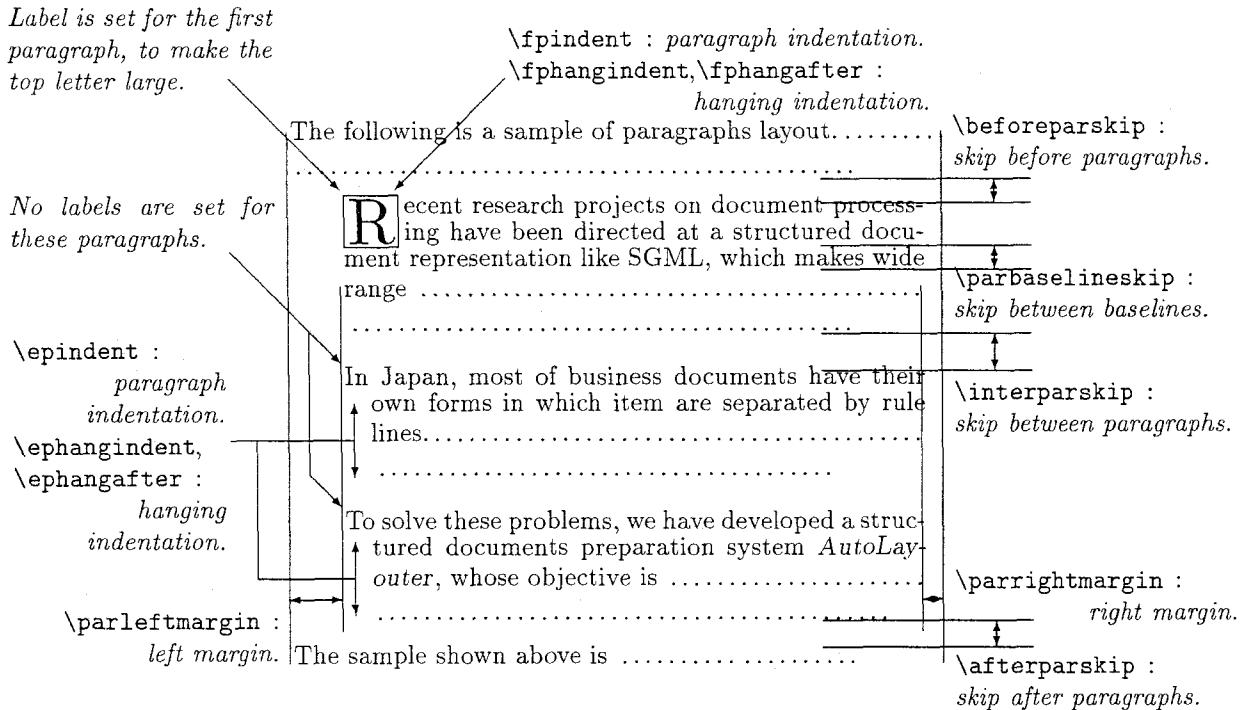


Figure 4: Paragraph Layout Model

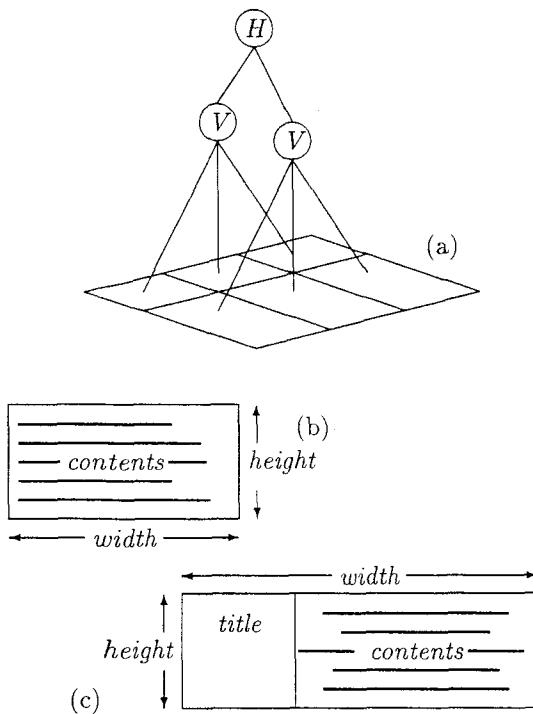


Figure 5: Form Layout Model

For these elements, we can put their contents into the main vertical list as they appear, using *paragraph layout*. In this case, assuming the name of the node is *foo*, mapping is performed simply by declaring the command `\node{foo}` and `\endnode{foo}` for paragraph layout. We call this *direct mapping*.

**Example 4:** Let us consider the case where one wants to define the layout of the structure element to provide an agreement style:

- A member should notify the consortium as soon as possible after modifying AutoLayouter.

Assume that the name of this structure element is "PROVISION". All that must be done is to specify the parameters to the paragraph layout model for PROVISION,

```
\par@node{def{PROVISION}%
  \fpindent{z0}%
  \afterparskip{.7ex plus .2ex}%
  \interparskip{.3ex plus .02ex}%
  {increment=1;ctrlayout=hang;}%
  {before=\bf{();after=})}}% counter
  {\% use default fonts
  {showctr}\% at the top of 1st paragraph
  {default}\% at the top of the others}
```

where `\par@node{def` is the command to define a structure element using the paragraph layout model. This definition directly maps the logical element "PROVISION" to its layout.

Most generic elements, such as itemizing, enumerating, and flushing, are also directly mapped with the `\par@nodedef` command.

*Indirect mapping.* In the case where the contents of each structure element are laid out irrespective of the order of their appearance, we can save the contents once and lay them out later. We call this type of mapping *indirect mapping*, and it applies to most forms, the *title* structure of `article`, and the *heading* of `letter`, for instance.

Now, let us consider this mapping with respect to macro definitions. Assume that an element *foo* is mapped indirectly, then the command `\node{foo}` should be defined with the form (see subsection Formatting Tree Structured Documents):

```
\def\node{#1#2{}...}
```

In the replacement text of this definition, argument #2, which contains a text segment, would be saved instead of being put out into the main vertical list. Only later would it be put into the main vertical list.

**Example 5:** Let us consider Example 3 again. *ALTEX's toolkit provides the command that directs an element to save the contents of a text segment using a macro definition. With this command, the node AUTHOR can be defined as:*

```
\def@nodedef{AUTHOR}{10}{}{}
```

where the first argument is the name of the element, the second argument specifies how many occurrences of the element can be allowed, and the last argument holds the initial value for the element.

For each occurrence of the element *AUTHOR*, `\beginnnode{AUTHOR}{...}`, `\endnnode{AUTHOR}` is expanded. In this expansion, the text segment is defined as the macros `\@AUTHORi`, `\@AUTHORii`, `\@AUTHORiii`..., and so on. The roman numerals *i*, *ii*, and *iii* in the name of the control sequences stand for the order of occurrence of the element.

Now, assume that *HEAD* is the parent node of *AUTHOR*, then one should define `\endnnode{HEAD}` as

```
\def\endnnode{HEAD}{...}
  \@AUTHORi
  ...}
```

in order to lay out the contents of the *AUTHOR* element.

## Conclusion

In this paper, we have described *AutoLayouter*, a structured documents preparation system that uses *TeX* and *LATeX* commands for structuring and formatting documents. By dividing a document structure into two layers, each of which contains logical

elements and generic elements, respectively, we can easily define the structure and layout of documents. Furthermore, we built-in an easy-to-use, dedicated user interface for editing the generic structure; this contributes to efficiency in document preparation.

In a future version, we plan to develop tools for defining the document's structure and layout, and also document management facilities.

## Acknowledgment

The authors would like to thank T. Ohno and R. Kurasawa, who developed Japanese *TeX*.

## Bibliography

- Adobe Systems Incorporated. *PostScript Language Reference Manual, Second Edition*. Reading, Mass.: Addison-Wesley, 1990.
- ISO 8879, "Information Processing—Text And Office Systems—Standard Markup Language (SGML)." Geneva ISO, 1987.
- Knuth, Donald E. *The TeXbook*. Reading, Mass.: Addison-Wesley, 1984.
- Kurasawa, Ryoichi. "Japanese *TeX* at ASCII Corporation" (in Japanese). Proceedings of *TeX* Users Group Japan, TX-97-5, September 1987.
- Kusumi, Yuki, Takashi Kakiuchi, Yoshiyuki Miyabe, and Kazu Tsuga. "Structured Document Preparation System *AutoLayouter*—Design and Implementation," IEICE Technical Report OS90-23, 1990.
- Lamport, Leslie. *LATeX: A Document Preparation System*. Reading, Mass.: Addison-Wesley, 1983.
- Miyabe, Yoshiyuki, Hiroshi Ohta, and Kazu Tsuga. "Structured Document Preparation System: *AutoLayouter*." *TUGboat*, 11#3, 353–358, September, 1990.

# Refining a Process

Linda Williams

The University of Tennessee Space Institute, Tullahoma, Tennessee, 37388 USA  
(615) 455-0631 x:233; FAX: (615) 454-2354  
bitnet: williams@utsiv1

## Abstract

Unlike word processing, the changes involved in using  $\text{\TeX}$  have not concerned the program, but have instead involved the type of user, equipment, and environment, all of which have evolved through the years and into the 90s. This paper profiles the various changes and offers suggestions for future structure and encouragement in the use of  $\text{\TeX}$ .

## Introduction

Before one can truly understand  $\text{\TeX}$  one must understand its original purpose and intended user, for these have impacted  $\text{\TeX}$ 's current use and future applications in ways perhaps not anticipated by Professor Knuth and his colleagues in the beginning. Over the last decade,  $\text{\TeX}$  has held its ground through numerous equipment and environment changes within both the scientific community and the computer industry. The direction and the problems surrounding the use of  $\text{\TeX}$  enter the conversations of computer experts and novice  $\text{\TeX}$  users alike. By shedding some light on  $\text{\TeX}$ 's history and by sharing insight and hindsight, the current and future use of  $\text{\TeX}$  can be brought into focus, along with what its proficient users consider to be its positive aspects and what novice or non-users consider to be its negative aspects.

## History

**Purpose and application.** As technology advanced into the computer age with its advanced mathematical capability, mathematicians became the forefathers of computer scientists. This group developed computers and numerous computer languages. By the late 70s, computers had advanced typesetting technology so quickly that within one generation we had gone from typewriting to  $\text{\TeX}$ !

For many years, the documentation of advanced technology was made available to the scientific community by an expensive and timely typesetting method that allowed little or no interaction with the originator of the documentation. More and more experimentation was being done on computer by the scientists themselves, but the documentation was still dependent on the old, traditional typesetting

procedures. There was an obvious need for a computer typesetting system that would enable its user to produce quality documentation.

This need was quite obvious to Dr. Knuth, as he started writing the many volumes of *The Art of Computing*. By the second volume, he had resolved to do the typesetting himself. With support from the National Science Foundation, Office of Naval Research, the IBM Corporation, the System Development Foundation, the American Mathematical Society, and Stanford University, he developed a program for typesetting his documentation, which he called  $\text{\TeX}$ . While refining  $\text{\TeX}$ , Dr. Knuth developed METAFONT, the Computer Modern fonts to be used with  $\text{\TeX}$ . Being a perfectionist, Dr. Knuth was not satisfied with the construction of the first Computer Modern fonts and called them Almost Computer Modern! The first version of  $\text{\TeX}$  was written in SAIL, not a widely used language since it only ran on DEC-20 computers. It was rewritten in Pascal and then in WEB to permit greater portability of the Pascal code. Others created a program to convert the WEB code into C code.

$\text{\TeX}$  was designed as a typesetting system to create beautiful mathematical and scientific documents.  $\text{\TeX}$  received instant acceptance by the scientific community. Documenting technology was no longer at the mercy of previous typesetting methods.  $\text{\TeX}$  enabled its original users to produce their own work and the results were as aesthetically pleasing as those achieved by the earlier costly and timely procedures. Finally, they had at their disposal a language that they could manipulate directly.

**Users.** The first users of  $\text{\TeX}$  were the initial programmers, a team put together by Dr. Knuth. As this team grew and expanded, it came to be a

group of very specialized, unique users. A story about this group may give a better picture: At the first organized meeting of *T<sub>E</sub>X* users, discussion centered around whether they should be an organized *democracy* or a loose *anarchy*; they chose the latter!

*T<sub>E</sub>X* enabled the typesetting of one's own documentation without encountering hassles with printers or publishers. *T<sub>E</sub>X* users became authors *and* editors of their own documentation. This was a *one-wizard* show: The user was keyboarder, typesetter, technical typist, technical editor, and proof reader; and if a new macro was needed, the same wizard wrote it. From fonts to drivers, the problems were handled; easily said, easily done—or close to it. (See Figure 1.) Computer scientists and mathematicians learned *T<sub>E</sub>X* with ease and excitement. (So as not to exclude other areas of expertise and interest, it should be noted that other divisions in academia, such as the English and history departments at various universities, soon tried their hand successfully at *T<sub>E</sub>X*.)

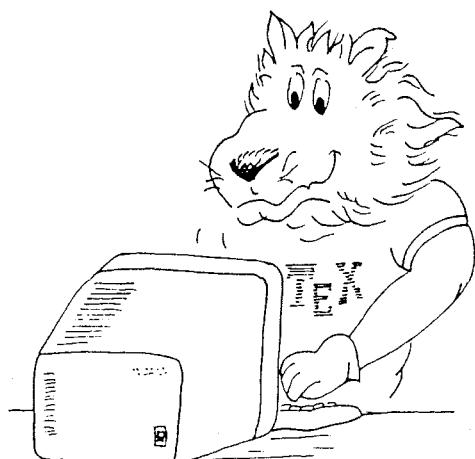


Figure 1: A One-Wizard Show

In the Preface to *T<sub>E</sub>X and METAFONT, New Directions in Typesetting* [1979], Dr. Knuth reversed a quote by Leonardo da Vinci, "Let everyone who is not a mathematician read my works." However, considering *T<sub>E</sub>X*'s original users, the original quote

by da Vinci, "Let no one who is not a mathematician read my works," should have been left alone. The original quote describes *T<sub>E</sub>X* and its wizards much more accurately. An even more accurate description of the wizard might be: "The trouble with having done something right the first time is that the wizard does not appreciate how difficult it is for anyone else."

As *T<sub>E</sub>X*'s popularity grew, so did the number of its users; and it established new typesetting standards for scientific and mathematical publications and documentation. The first users were the *T<sub>E</sub>X* pioneers, who were specialists in their fields. However, as with any new technology, the use and users changed with time and organization.

## Present

The *T<sub>E</sub>X* program is in the public domain. Dr. Knuth spent thousands of hours to make sure that "... the system would produce essentially identical results on all computers" [1990]. There are 1536 institutions and 3298 individual users of *T<sub>E</sub>X*.† *T<sub>E</sub>X* is used for all major European languages, and for others that are written either horizontally or vertically [Beebe 1990]; in more than 51 countries, the majority typeset English.† There are many publications that demonstrate and document *T<sub>E</sub>X*'s various and diverse applications and users.

**Current applications.** Current applications are numerous. Aside from extremely specific applications, often demonstrated and published in the *TUGboat*, the primary application is still to typeset scientific and technical documentation and to solve difficult formatting problems.

The academic environment revolves around publications. Institutions are frequently evaluated in terms of their publications. The funding of many organizations depends heavily on presentations and documentation. As a result, an increasing number of journal and other publishers use *T<sub>E</sub>X* and/or accept submissions in *T<sub>E</sub>X*. Many government and government subcontractors use *T<sub>E</sub>X* exclusively to typeset technical documentation and publications [McCaskill 1988].

**Users.** The users of these various applications fall into two categories: (1) the do-it-yourself wizards, and (2) the multilevel document-preparation-system team members. (This second group must include at least one person who will be responsible for *T<sub>E</sub>X*

---

† This information was obtained from *T<sub>E</sub>X* Users Group, May 1991.

instruction and system language support [Gibson 1990].)

The first group is similar to the TeX pioneer; however, the use of TeX in a one-person operation is no longer necessarily by choice, but is influenced by time constraints, level of expertise, and funding.

The structure of the second group can be as simple as two people, perhaps one author and one typesetter, or as complicated as eight individuals, each of whom does only one of the eight various tasks involved in document preparation. The number of personnel doing these tasks vary and responsibilities overlap in some organizations, depending once again on time constraints, level of expertise, and funding. The roles that must be filled are: author (for text), technical editor, design editor, illustrator (for graphics), typesetter, keyboarder, proofreader, and printer/photocopier. (See Figure 2.)



Figure 2: Multilevel Complexity

The ability to work together toward a common goal is fundamental to the refinement of any process. If an organization has two or more people involved in the various steps of document preparation, each member's understanding and knowledge of TeX and of document preparation may differ widely, but all must work well, together as well as independently.

**Observation — filling in the gaps.** It is not difficult to outline TeX's current applications and

users. In fact, one sentence summarizes this observation: TeX's most efficient and effective use is to support technical documentation departments at educational institutions, research organizations, government agencies, and publishing companies.

How the typical TeX user moved from being an individual user to being a member of a highly specialized team of technical users and support personnel was less subtle and organization-dependent. Various factors impacted these changes: time constraints, computer expertise, and funding. Historically, organizations and institutions that implemented TeX as soon as it became available on their computer systems later experienced structural changes. However, TeX still addressed the majority of their typesetting problems, was in the public domain, and produced beautiful documents in a reasonable length of time.

At TeX's advent, word processing software was not as user friendly as it is today, and TeX could be used to solve nearly every typesetting problem. However, TeX use was not limited to wizards. What could be so difficult about using a computer language to typeset everything? The answer became apparent when avid TeX supporters and users wanted (or needed) to rely on clerical staff to typeset technical documentation. TeX's high learning curve became apparent and the need for TeXnical support became quite obvious: The underpaid, overworked, stressed-out, clerical support staff emitted cries of frustration, while the technically-oriented document personnel emitted cries of gratitude. The positive and negative aspects of TeX appeared all at once, all involving accessible (at various user levels) information, technical support, and structured organizational levels (or the lack thereof). At this point, WYSIWYG word processing systems for use by non-technical clerical staff came of age, and TeX was reclaimed by those who needed it *and* could use it effectively and efficiently.

Various organizations have flip-flopped from word processing packages to TeX or from TeX to word processing packages [Hoover 1989]. Conscientious institutions utilize both systems according to their typesetting requirements. Time constraints, computer expertise, and funding are now factors that organizations can analyze to determine the best possible cost-effective document-preparation system for meeting their needs. Organizations that previously relied solely on TeX can now restructure. By placing their capable TeXnical personnel where they will be of greatest benefit to the entire system, that is, in a technical documentation department,

and using word processors for non-technical uses, they can better use their often-limited resources.

## Future

**TEXnically speaking.** The future of TEX depends on its ability to meet the varying and continuously growing needs for the typesetting of technical documentation. This is not for a novice, like myself, to speculate on what technical innovations need to be addressed; excellent observations have already been presented by Nelson Beebe [1990] and Frank Mittelbach [1990].

Today's market is flooded with word processing software that address most typesetting and formatting requirements but that cannot typeset difficult technical, scientific, and mathematical documentation. As word processing software continues to address the needs of the commercial industry, TEX must also adapt and integrate and, beyond this, again set new standards and goals.

**Non-TEXnically speaking.** There are several areas of promotion and successful marketing and development strategies that TEX users and supporters have failed to use; the leaders of TEX need to address these. They include: encouraging more-accessible written information to close the gaps between user levels, such as dictionaries containing computer- and TEX-user terms; providing multi-level computer-dependent and TEX-related encouragement and publications; advertising already-established publishing practices; giving more than lip service to suggestions; and making sure that distributed information is received, is understood, and is applicable. The basic idea must be to establish TEX's uses and users, and to support them.

## Conclusion

For hundreds of years, society advanced technologically through the sharing of scientific knowledge. This century has seen many technological advancements become commercial interests, to the point that commercial interests too often dictate the progress of technology. It has been difficult for TEX to hold to the ideal of shared knowledge in the face of commercial exploitation, but it is this ideal that has made TEX valuable to computer science and to the documentation of scientific information. In short, TEX is a brilliantly written, designed, and executed program that was far ahead of its time. If it had been developed later, TEX could perhaps

have been more easily adapted and perhaps the original goals would have been different. However, it is the continuing ability of TEX users to use this hindsight to their advantage, along with their willingness to solve and share technical and non-technical problems and solutions, that makes the use of TEX such a refined process. Whatever the future holds for TEX, there is no doubt that it has already passed the test of time.

## Bibliography

- Beebe, Nelson. "Comments on the Future of TEX and METAFONT," *TUGboat* 11#4 (November 1990), page 490–494.
- Gibson, Helen. "A Noddy's Guide to using TEX for Text Production: From Manuscript to Bromide," *TUGboat* 11#3 (September 1990), pages 393–399.
- Guenther, Dean, Ph.D. Washington State University, Personal Interview on May 1990.
- Hoover, Anita. "Using WordPerfect 5.0 to Create TEX and LATEX Documents," *TUGboat* 10#4 (December 1989), pages 549–559.
- Knuth, Donald. "The Future of TEX and METAFONT," *TUGboat* 11 #4 (November 1990), page 489.
- Knuth, Donald. *TEX and METAFONT, New Directions in Typesetting*. Bedford, Mass.: American Mathematical Society and Digital Press. 1979.
- Lafrenz, Mimi. "Textbook Publishing—1990 and Beyond," *TUGboat* 11#3 (September 1990), pages 413–416.
- Martin, Charles. "TEX for TEXnical Typists," *TUGboat* 11#3 (September 1990), pages 425–428.
- McCaskill, Mary. "Producing NASA Technical Reports with TEX," *TEXniques*, 7 (August 1988), pages 1–10.
- Mittelbach, Frank. "E-TEX: Guidelines for Future TEX Extensions," *TUGboat* 11#3 (September 1990), pages 337–345.

# A Text Processing Language Should be First a Programming Language

Luigi Semenzato, Edward Wang

Computer Science Division, University of California, Berkeley, California, 94720

Internet: luigi@ginger.Berkeley.EDU, edward@ucbarpa.Berkeley.EDU

## Abstract

Historically, typesetting languages have been designed for the entry of text. An embedded command language has since become important, indeed essential, but has remained a second-class citizen, sometimes masquerading as text, invariably clumsy and inadequate. We have designed a language that is a full-function programming language *with embedded text*. This shift in emphasis results in a level of consistency, flexibility, and power not otherwise possible.

## Introduction

A batch-style computer typesetting system accepts text files as input, to produce formatted documents as output. Most such systems are extensible. They allow definitions of new document styles and commands. Some, like *TeX*, also allow the input syntax to be changed. To do all this, the format of the input must be a complex language. The design of this language affects the robustness, ease of use, and overall quality of the whole system.

A document, therefore, is a mix of text and commands, some of which define new commands or make syntax changes. Existing systems have emphasized the text portion of the input. In these languages, the commands are an afterthought. They often follow the inconvenient lexical conventions of the surrounding text, and make awkward programming languages. This paper describes our attempt to reach a better design, by turning the traditional language inside out, giving priority to commands and programming. We call this system and its language *Aleph*.

An Aleph document is a sequence of commands, some with embedded text as arguments. The commands are in a programming language with a fixed syntax. Text, on the other hand, can have a user-specified syntax. Each command builds an internal representation of a portion of the document. This representation is then processed to produce the output.

Aleph is an evolving design. Its current realization (sometimes called Aleph<sub>0</sub>) is written in Lisp. Our immediate goal is not to produce a complete typesetting system, but to design a language that is a tool for both writing the system and using it.

One consequence is that the Aleph system does no actual typesetting, but generates *TeX* as output.

The sections of this paper describe selected aspects of Aleph, in this order: basic constructs, extensible syntax, internal representation, implementation. The rest of this introduction is a discussion of some of the issues in typesetting-language design.

**Syntax separation.** Commands should not obey the syntax of the text around it.<sup>1</sup> For example, it is often convenient to ignore whitespace and line boundaries in a program, but not always possible in the text of a document. In *TeX*, it is sometimes hard to predict whether spaces and newlines in and around commands will be part of the output. User-defined syntax is a useful feature, but exacerbates the problem—commands that change the syntax may affect themselves.

In Aleph, commands (both definitions and invocations) are in a language with a fixed syntax, while embedded text follows a different set of rules. Syntax changes for text are well supported.

**Programming.** Extensibility is a very desirable feature in a batch typesetting system. It should be supported with a full-function programming language.

Extensibility is essential if new document styles are to be written, and in practice, all but the most casual users define shorthands for frequently used text and command sequences. For the latter, a macro language is the natural choice—after all,

<sup>1</sup> We use the words *lexical* and *syntactic* interchangeably, partly because text-processing languages have little of what can be called syntax, but mostly because *lexis*, a candidate counterpart for *syntax*, is not a common computer-science term.

nothing is easier to understand than textual substitution. Indeed, existing systems have preferred macro languages over more procedural ones. On the other hand, a document style is a large program. L<sup>A</sup>T<sub>E</sub>X, for example, is 2000 lines of code. Real programs need real programming-language features. T<sub>E</sub>X, for one, has conditionals and loops, but no real data structures or indeed any support for writing large programs. In addition, macros themselves become unwieldy in large programs. That T<sub>E</sub>X allows fine control over macro expansion is an indication of its complexity.<sup>2</sup>

Intertwined with issues of linguistic power is the fact that typesetting systems are always *implemented* in one language (a general-purpose programming language) while they *implement* another. (Most complete systems, of course, are written in both.) This practice limits the power of user-written programs—when a primitive to do something does not exist, it cannot be done. The existence in T<sub>E</sub>X of complex functions as primitives (such as \halign) may be an instance of this.

Aleph is a full-function programming language, with data types to represent textual objects and functions to manipulate them. Users at all levels use the same language. There is no barrier between what the user can do and what the system can do.

## Aleph and Lisp

Aleph is embedded in Common Lisp. In other words, Aleph is implemented in Lisp as a set of functions, data types, and syntax extensions. An Aleph programmer must use at least as much Lisp as Aleph.

Lisp is an expression language. Every program construct is a value-producing expression called a *form*. A function-call form is surrounded by parentheses: (f 1 2 3). Here, f is the name of the called function. It is passed three arguments: 1, 2, and 3. Identifiers like f are called *symbols*. In this paper, a symbol can be any sequence of letters and -s. A symbol in the first position of a function-call form is a function name. A form that is a symbol alone is a variable. The form (f x y z) calls f with the

<sup>2</sup> Macros are not inherently less powerful. After all, we know that lambda calculus is turing-complete. T<sub>E</sub>X's own linguistic problems are also quite complex. They are in part due to the need to delay execution in some situations. In any case, complexity is perhaps not a deadly sin, but the apparent unpredictability that comes with complexity is.

values of variables x, y, and z. Forms can be nested: (f 1 (g (h 2) 3) 4).

Common Lisp also has characters and strings. A string is enclosed in double-quotes: "in double quotes". A character is written with the prefix #\. For example, #\a is a, #\% is %, and #\\ is \.

A symbol that begins with a colon, :, is a *keyword*. A keyword is an uninterpreted identifier that stands for itself. It is used like the identifiers defined by an enumerated type in C or Pascal.<sup>3</sup>

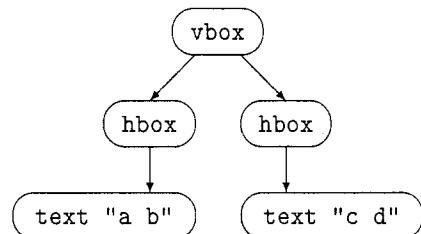
Not all forms in parentheses are function calls. There are built-in and user-defined forms that have special syntax (nevertheless made out of symbols and parentheses), and interpret arguments in special ways. The most visible ones in Aleph are those that begin with def.

We now know enough Lisp to understand the Aleph extensions.

A document (or a part of a document) in Aleph is represented by a tree, like nested T<sub>E</sub>X boxes and lists. For example, the T<sub>E</sub>X box of boxes made by

```
\vbox{\hbox{a b}\hbox{c d}}
```

would have a fairly similar Aleph tree:



Trees are constructed using *tree-building functions*—Lisp functions that create tree nodes. The last example is constructed by the form

```
(vbox (hbox (text "a b"))
      (hbox (text "c d"))))
```

An Aleph document is just a sequence of such tree-building forms. However, entering a large document with nested forms is rather clumsy. For most forms, there is an equivalent *Aleph string* that is more concise.

An Aleph string (or just *string*, when confusion with Lisp string is unlikely) is enclosed in brackets: [ and ]. For example, [some text] is equivalent to (text "some text"). As in T<sub>E</sub>X, newlines and tabs in Aleph strings are treated like spaces, and consecutive spaces are treated like

<sup>3</sup> If this is confusing, then just treat keywords as strings—think "xyzzy" when you see :xyzzy. Keywords have no meaning except in their name and in their use.

one. `[some\_text]` is not the same as `(text "some\_text")`.

The equivalence between a string and its corresponding form is strict. The string actually *becomes* the form as it is read by Lisp. The rest of the Lisp system never sees Aleph strings.

Since they are equivalent, strings and forms can be mixed freely. We now know enough to enter a simple document:

```
(par [A very short document of
      a single short paragraph
      of a middling sentence.])
```

Just as we can go from Lisp to string, we can go from string to Lisp. The string

```
[an @it [italicized] word]
```

has a string nested in a Lisp form that is in a string. It is equivalent to

```
(group (text "an ")
       (it [italicized])
       (text " word"))
```

which is in turn equivalent to

```
(group (text "an ")
       (it (text "italicized"))
       (text " word"))
```

Since this string-Lisp-string double take is so common, we have defined a shorthand for it:

```
[an @it[italicized] string].
```

The escape character, `@`, is very much like `\` in TeX. A number of `@`-triggered features are defined in Aleph, and the user can define more. This and other forms of user control over strings are the subject of the next section.

## Mode and Syntax

A *mode* governs the way Aleph strings are turned into tree-building forms. In TeX, the equivalent concept is implicitly defined by the catcodes. Aleph, on the other hand, supports a data type, *mode*, that encapsulates all the information that defines a mode.

For example, to define a mode in which the character `%` expands to the italicized word “*Aleph*,” we would write

```
(defsyntax aleph ()
  (#\% (it [Aleph])))
(defmode aleph aleph)
```

The first statement creates a new syntax table, `aleph`, with the character definition. The second statement creates the new mode, also named `aleph`, that uses the new syntax (named by the second `aleph` on the line). (We often, but not always, use the same name for a mode and its syntax.) The

new mode can now be invoked using an escape sequence:

```
[... @@aleph[% is embedded in Lisp]...].
```

We can also give `aleph` a pair of delimiters:

```
(defmode aleph aleph
  :open #\{
  :close #\}),
```

and use them to invoke the mode more concisely:

```
[... { % is embedded in Lisp } ...].
```

This is one of the reasons for separating `defmode` and `defsyntax`. A syntax is the character definitions used by a mode. The mode itself uses a syntax, but may also have some supporting attributes.

A syntax can be built on top of an existing syntax (assuming we already have a `verbatim` syntax defined):

```
(defsyntax valeph (verbatim)
  (#\% (it [Aleph])))
(defmode valeph valeph)
```

Thus, `valeph` has the behavior of `verbatim` but also recognizes `%`.

A syntax can be a combination of others. We could have (and indeed should have) defined `valeph` like this:

```
(defsyntax valeph (aleph verbatim))
```

The syntaxes in Aleph form an inheritance hierarchy. Each syntax definition specifies a list of parent syntaxes (multiple inheritance) and some local additions. Looking up the definition of a character in a syntax is a matter of trying, in order and until a definition is found, the local definitions and then the parents (left to right). In each parent, the same process is repeated.

When modes nest (such as in `[...{...}...]`), the lookup is first done in the closest enclosing mode, then repeated in surrounding modes (inside out), until a definition is found. Inside `[...{ % is embedded in Lisp } ...]`, the definition for `%` is found in mode `aleph`, but the other characters behave as they would outside the braces. This nesting is lexical, even when a string goes in and out of Lisp: `[...{@(it [%...])}...]`.

The full form of `defsyntax` looks like this:

```
(defsyntax <name> (<parent>...)
  <default-definition>
  (<chars> <definition>)
  ...)
```

`<Chars>` is either a single character or a Lisp string representing a set of characters. `<Definition>` is the definition given to the character or characters. Any number of `(<chars> <definition>)` pairs can be specified. Characters not explicitly mentioned receive

$\langle default-definition \rangle$ , which can be left out, to leave them undefined. So far, we know a character definition can be a Lisp form. It can also be one of several keywords, some of which we will see later. In the most extreme case, a definition can be a Lisp function. We won't use any of these in this paper.

A syntax or mode can be changed: character definitions and parents can be added and deleted; modes can lose or gain delimiters.

The basic Aleph defines these modes:

```
(defsyntax delimiter ())
(defsyntax escape ()
  (#\@ :escape))
(defsyntax standard ()
  (#\Newline :space)
  (#\Space :space)
  (#\Tab :space)
  ... more definitions ...)
(defsyntax default
  (delimiter escape standard))
(defsyntax group ())
(defmode group group
  :open #\[ 
  :close #\])

```

Default is the outer-most syntax of all Aleph strings. Escape contains the single character `@`. Delimiter contains the delimiters defined with defmode. Standard is the rest of the definitions for the default mode. Group defines no characters. It is the syntax for the delimiters [ and ]. Delimiter is initially empty, but (defmode group ...) soon adds two definitions to it.

Escape, delimiter, and standard are separate syntaxes to allow modes to inherit them independently. For example, one may wish to define a mode that behaves like the LATEX verbatim mode but also recognizes the escape character:

```
(defsyntax weak-verbatim
  (escape verbatim))
```

This approach allows a change to the escape character to be effective everywhere.

The escape character behaves like a mode, but without a fixed closing delimiter. The dispatch syntax controls escape-sequence processing. These escape sequences are supported:

`@(...)`

This is the escape into Lisp we have seen. The Lisp form should be a tree-building form.

`@(symbol)`

This is equivalent to `@(symbol)`.  $\langle symbol \rangle$  must be a reasonable-looking Lisp symbol (made out of letters and -s).

`@(symbol)⟨delimited-string⟩...`

If the `@(symbol)` sequence is followed immediately by an opening delimiter (defined in syntax delimiter), then the delimited string becomes the argument of  $\langle symbol \rangle$ :

`@(⟨symbol⟩ ⟨delimited-string⟩)`

$\langle Delimited-string \rangle$  can be repeated any number of times. For example, `@f[Aleph][Beth]` is the same as `@(f [Aleph] [Beth])`.

`@$(symbol)⟨open-delim⟩⟨text⟩⟨close-delim⟩`

Enter mode  $\langle symbol \rangle$  for the duration of  $\langle text \rangle$ .  $\langle Text \rangle$  can contain any character other than  $\langle close-delim \rangle$ .  $\langle Open-delim \rangle$  is any character.  $\langle Close-delim \rangle$  is ), ], }, or >, if  $\langle open-delim \rangle$  is (, [, {, or <, respectively. Otherwise,  $\langle close-delim \rangle$  equals  $\langle open-delim \rangle$ . This is how modes without delimiters are invoked.

`@;`

The rest of the line, including the end-of-line character, is ignored.

`@⟨accent⟩`

A number of accents are defined in Aleph.

`@\⟨char⟩`

The character  $\langle char \rangle$ .

`@⟨char⟩`

This is equivalent to `@\⟨char⟩`, if  $\langle char \rangle$  has no defined behavior (one of the above).

**Flexibility: mechanism and policy.** The user of a mode is not necessarily the writer of the mode. This is particularly true when canned Aleph code from a library is used. LATEX, for example, has such a library. When a mode or a syntax is to be reused, the programmer must anticipate the possible uses and choose the implementation accordingly. To do this requires some skill, but also a flexible syntax mechanism.

For example, the mode `aleph`, though frivolous, belongs to a common class of user-defined modes. It defines only a few characters, so must be used in conjunction with another mode (if nothing else, with `default`). We expect such a mode to be used in several different ways, depending on the user's needs:

- Enter mode when necessary, using delimiters or `@$`.
- Use everywhere, by making it a parent of `default`. An Aleph function is provided to do this.
- Combine with other modes to make new ones (like `valeph`).

As we have seen, the definition of `aleph` does allow this freedom.

**Consistency.** A mode like LATEX's `verb` is very easy to define in Aleph:

```
(defsyntax verb () :char
  ;; make newline act like a space too
  (#\Newline (text " ")))
(defmode verb verb)
```

All characters are given the definition `:char` (meaning just the character itself). Using `verb` looks like its LATEX counterpart: `\$verb|...|`.

It is simple to define `verb` in Aleph — we do not have to write catcode-changing macros. Its other advantage is the consistency of behavior. Wherever `\$` is recognized, `\$verb|...|` can be used. Unlike in LATEX, there are no unpleasant surprises depending on context or content.

## Trees

As mentioned, an Aleph program constructs a tree that represents the document internally. Nodes in the tree have a *type* that indicates what object each node stands for. The type is named by a Lisp keyword. For instance, a node of type `:box` represents a box, and its children the content of the box; a node of type `:par` represents a paragraph, and its children text or other material that needs to undergo line breaking. We have given examples of how to construct such trees in earlier sections.

A tree fully specifies a document fragment, but requires some processing before it can be used for output. Aleph performs such processing in a *traversal* pass.

Aleph provides a number of primitive node types. One can also define new types in the following way:

```
(defnode <type>
  :constructor <c-function>
  :traverse-function <t-function>
  :output-function <o-function>)
```

Here `<type>` is an arbitrary keyword denoting the node type. `<C-function>` is a function that constructs a node of that type. If one is not supplied, a standard constructor is provided, with a name equal to the node type (without the colon). `<T-function>` is the traversal function for nodes of this type. `<O-function>` is called during a similar traversal to output the document.

Each instance of a node has an associated set of named values called *attributes*. Attribute names are also Lisp keywords. For instance, a `:box` node has a `:direction` attribute indicating if its components should be stacked horizontally or vertically; a `:par` node has a `:width` attribute, whose numeric value selects the width to be used for line breaking.

A few attributes are assigned at node construction time. Other attributes represent printing information, such as the final position and size of the formatted object. These attributes are filled in by the tree traversal. This starts at the root of the tree and proceeds by calling the traversal function of each node it visits. Besides computing attributes, traversal functions are also allowed to modify the tree locally.

To clarify these concepts, we introduce a simple example. We add the node type `:f-box`. This node has a single child representing some printable object. If the width of the object is less than 1 inch, it is printed centered in a 1-inch horizontal space; otherwise three dollar signs are printed.<sup>4</sup>

```
(defnode :f-box
  :traverse-function #'trav-f-box)
```

The traversal function for `:f-box` is `trav-f-box`; its output function is the default output function, which just outputs the node's children.

```
(defun trav-f-box (n)
  ;; First visit the (only) child
  ;; of this node.
  (traverse (child n))
  ;; Then destructively modify this node.
  ;; Change its type:
  (setf (type n) :box)
  ;; Specify the width:
  (setf (attr :width n) !1inch)
  ;; Change its child:
  (setf
    (child n)
    ;; Use a centering construct
    (center
      (if (< (attr :width (child n))
              !1inch)
          ;; and inside it, put
          ;; either the old child
          (child n)
          ;; or three dollar signs.
          [$$$]))
    ;; Traverse the modified node
    ;; to set the glue.
    (traverse (child n))))
```

This example contains a few unfamiliar but quite simple Lisp and Aleph constructs:

- the `defun` form defines a Lisp function named `trav-f-box`, that takes the single argument `n` and operates on it;

---

<sup>4</sup> The letter *f* in `f-box` stands for FORTRAN.

- the Aleph form (`child x`) refers to the value of the single child of `x`, and the form (`attr name x`) refers to the attribute `name` of node `x`;
- `setf` is the Lisp assignment operator. (`(setf place value)` replaces the old value of `place` with `value`. So `(setf (child n) ...)` replaces the child of `n`;
- `!(number)<unit>` is Aleph's way of specifying a length;
- `center` is an Aleph function that returns a group with appropriate glue for centering.

This example reveals that our typesetting primitives are very similar to those of `TeX`. In fact, we think that most of `TeX`'s primitives are well designed and we are not attempting to improve on them.

One should define new node types with their own traversal functions only when direct access to the typesetting engine is needed. We expect style writers to be able to do most of their programming at the level of mode definition and tree construction. The system programmer (us) should provide enough node types to satisfy the most common needs.

## Current Status and Future Directions

As we are submitting this paper, the implementation of Aleph contains the described syntax mechanisms and intermediate representation. We have also defined a small number of node types, most notably paragraphs, boxes, and glue. The output routines produce plain `TeX`. `TeX` is also used in interactive mode to perform some computations currently not implemented in Aleph, such as finding the widths of objects in our table constructor. The Aleph process communicates with the `TeX` process through Lisp streams connected to a UNIX socket pair.

Aleph relies on `TeX` for ligatures, line breaking, math, and output. As a consequence, we expect the exact semantics of traversal and retraversal to evolve, as more is demanded of them. Also, it is at present difficult to estimate the system's efficiency, though we believe the tree-and-traversal model is not fundamentally inefficient.

Of the missing features, ligature and math are perhaps the hardest for our model. We plan to tackle them first. Unrelated to `TeX`, we are also considering ways to extend the syntax mechanism to recognize multicharacter sequences.

Aside from completing this implementation and refining it into a practical tool, our work suggests many other research directions. For instance, to what extent is Aleph's intermediate representation suitable for a WYSIWYG-style document editing, with incremental processing? And if it is, would it

simplify the task of integrating programmatic and WYSIWYG interfaces? We have not tried to answer these questions, but we hope that our work, by allowing one to look at an old problem in a new way, will provide both a stimulus and a vehicle for further research.

## Acknowledgments

We thank Ethan Munson for his useful suggestions. Luigi Semenzato did part of this work at the Dipartimento di Informatica, Università di Padova, Italy.

## Bibliography

- Steele, Guy L. *Common Lisp: the Language*. Burlington, Mass.: Digital Press, 1984.
- Kernighan, Brian W. "Issues and Tradeoffs in Document Preparation Systems." Pages 1–16 in *Proceedings of the International Conference on Electronic Publishing, Document Manipulation & Typography*, R. Furuta, ed. Cambridge: Cambridge University Press, 1990.

## Appendix

### Table example

This is an example of use of our table primitive, with the resulting output. The `table` constructor is a Lisp macro. Macros are a powerful feature of Lisp that we cannot attempt to explain here. In this context, just think of a macro as a function with a more flexible argument-passing mechanism.

```

;;; We thank Marcia Feitel for correcting an important omission.

(line
  (center (bf (bind :size 12 [From page 236 of the TeXbook, more or less]))))

(vskip !0.5in)

(line
  (center
    (table
      ;; Half of the padding goes before the column, half after the column.
      :pad !0.5cm
      ;; The vertical padding goes between rows.
      :vpad !2pt
      ;; The template is a list of column descriptors.
      ;; Each descriptor is a function, or a list of functions,
      ;; called in turn with each corresponding entry in a row
      ;; as argument.
      :template ((right bf) (center it) center center left)
      :rows
      ;; These are the rows. Each row is a list of entries.
      (((sl [American]) (sl [French]) (sl [Age]) (sl [Weight]) (sl [Cooking]))
       ((sl [Chicken]) (sl [Connection]) (sl [(months)]) (sl [lbs.])
        (sl [Methods])))

      ;; A special row that spans all columns.
      (:span-all (left (vbox [] !0.1in)))

      ;; $ is the Aleph delimiter for the tex-math mode, an escape into TEX's math mode.
      ([Egg] [Oeuf] [-2\over3] [{1\over6}] [Boil, Fry, Poach, Raw])
      ([Squab] [Poussin] [2] [{3\over4} to 1] [Broil, Grill, Roast])
      ([Broiler] [Poulet Nouveau] [2 to 3] [{1\over2} to {2\over1}])
      [Broil, Grill, Roast])
      ([Fryer] [Poulet Reine] [3 to 5] [2 to 3] [Fry, Saut@'e, Roast])
      ([Roaster] [Poularde] [{5\over1}\over2] [Over 3] [Roast, Poach, Fricassee])
      ([Fowl] [Poule de l'Ann@'ee] [10 to 12] [Over 3] [Stew, Fricassee])
      ([Rooster] [Coq] [Over 12] [Over 3] [Soup stock, Force meat])))))

```

## From page 236 of the TeXbook, more or less

American Chicken	French Connection	Age (months)	Weight lbs.	Cooking Methods
Egg	<i>Oeuf</i>	— $\frac{2}{3}$	$\frac{1}{6}$	Boil, Fry, Poach, Raw
Squab	<i>Poussin</i>	2	$\frac{3}{4}$ to 1	Broil, Grill, Roast
Broiler	<i>Poulet Nouveau</i>	2 to 3	$1\frac{1}{2}$ to $2\frac{1}{2}$	Broil, Grill, Roast
Fryer	<i>Poulet Reine</i>	3 to 5	2 to 3	Fry, Sauté, Roast
Roaster	<i>Poularde</i>	$5\frac{1}{2}$ to 9	Over 3	Roast, Poach, Fricassee
Fowl	<i>Poule de l'Année</i>	10 to 12	Over 3	Stew, Fricassee
Rooster	<i>Coq</i>	Over 12	Over 3	Soup stock, Force meat

# Should T<sub>E</sub>X be Extended?

Michael Vulis

MicroPress Inc, 68-30 Harrow Street, Forest Hills, New York, 11375 USA  
718-575-1816; FAX: 718-575-8038  
Bitnet: cscmlv@ccnyvme

## Abstract

This article examines three problems discussed in recent issues of TUGboat: Graphics inclusion, Font rotation, and Font selection scheme. The author compares the traditional solutions to the problems (pure T<sub>E</sub>X) to the solutions that can be obtained by slight extensions to either the T<sub>E</sub>X language primitives or the driver programs. For each problem, the article shows what can and cannot be achieved with puristic (Clean) T<sub>E</sub>X solutions; it will also describe how the limitations can be overcome with (Dirty) T<sub>E</sub>X language extensions and document the extensions.

## On T<sub>E</sub>X

Since its inception eleven years ago, T<sub>E</sub>X has remained essentially unchanged. Meanwhile, the world of personal computing has advanced dramatically.

Circa 1980, a personal computer with 64k RAM was still considered advanced. Laser printers did not exist. WordStar and DisplayWrite were leaders in word processing. T<sub>E</sub>X was a revolution.

Circa 1985, PostScript was around, but prohibitively expensive. Proportional fonts were still a novelty. Desktop publishing was yet non-existent. Graphics was non-integratable. And T<sub>E</sub>X shined.

Circa 1990, leading word-processors (i.e., WordPerfect) format text almost as well as T<sub>E</sub>X, and perhaps easier. They handle graphics and tables much better than T<sub>E</sub>X, they generate indices and they spell check. They do not handle equations as well as T<sub>E</sub>X; however, they are not far off.

Circa 1995, T<sub>E</sub>X could become a historical curiosity.

## On Extensions

Software systems that remain unchanged are destined for oblivion. T<sub>E</sub>X has lasted this long primarily because of its fresh start: immense superiority of T<sub>E</sub>X over other typesetting systems. This superiority is over, or almost over. To survive, T<sub>E</sub>X needs to evolve.

There are two ways the evolution of T<sub>E</sub>X can proceed: either one person, possibly even the Grand Wizard himself, can undertake serious and continuing rewriting of the system, or this rewriting

will be done in possibly incompatible ways by several implementors. Since the Grand Wizard has declared his unwillingness to make any changes in the design, the second possibility appears likely. The goal of the T<sub>E</sub>X community should be to ensure that this rewriting does not get out of hand—to define the process of directing, implementing, documenting and *sharing* the extensions.

Historically, language compatibility has been assured by language standards. The existence of Standard (ANSI) Pascal, in particular, made T<sub>E</sub>X itself possible. A starting point, therefore, can be defining Standard T<sub>E</sub>X (T<sub>E</sub>X3.14159). T<sub>E</sub>X3.14159 will be identical to the T<sub>E</sub>X appearing in *The T<sub>E</sub>Xbook*, with the following *change*: it will implement integer register compatibility. A ‘T<sub>E</sub>X’ can be deemed to be a ‘T<sub>E</sub>X’, if any source file that either starts with

```
\compatibility=0
\let\compatibility\undefined
```

or does not include any of the new keywords should be handled identically by this T<sub>E</sub>X and T<sub>E</sub>X3.14159. Notice that this definition both supersedes the TRIP compatibility test and ensures that T<sub>E</sub>X documents can stay compatible between different systems.

## On This Paper

With this definition in mind we will proceed with the study of a few changes to T<sub>E</sub>X that implement some of the desirable extensions. While the size of this paper will prevent us from presenting complete changes to the T<sub>E</sub>X code, these are available from the author (requestware). The extensions described

in this paper were implemented and tested under VTeX system (see *TUGboat*, August 1990). The four extensions discussed here include:

- Font rotation
- Incorporation of graphics
- Automatic indices
- Font selection and/or substitution

### Case Study I: Font Rotation in TeX

Of the three problems discussed in this article, font rotation probably received the least attention. The reason for it may be that before June 1990, no one has realized it was possible and afterwards no one thought it was practical. In June 1990, Alan Hoenig opened the chapter on Font Rotation with his beautiful examples (see *TUGboat*, 1990 Conference Proceedings) and closed the chapter with a scary explanation of how they were made.

Hoenig's approach consists of generating a series of TeX fonts via METAFONT, one font per required angle of rotation. For instance, to typeset a 24-character line of text around a circle, one would need to generate 24 variants of the original font. Similarly, a 100-character example requires 100 pre-generated fonts, 101-character example requires 101 different fonts ( $gcd(100, 101) = 1$ ), while a 300-character can use the fonts generated for the 100-character example, but cannot be printed in most TeX versions ( $\text{font\_max} \leq 255$ ). Hoenig's use of METAFONT was forced by two distinct reasons: drivers' inability to rotate fonts and, more to the point, TeX's inability to position characters when typesetting not on a horizontal line. To correctly update the reference point, TeX needs to know the sine and cosine of the typesetting angle; Hoenig made METAFONT compute them and pass them to TeX as extra `\fontdimen` parameters.

Hoenig's examples remain in the realm of curios, since it would not be practical to generate many fonts each time rotation is required. Even when drivers support font rotation (VTeX drivers do and PostScript drivers can), the problem remains as to how to compute sine's and cosine's. While it can be proven that macros for computing trigonometric functions can be written in TeX, a somewhat easier (and much faster) way is to simply transplant the relevant code (the `n_sin_cos` procedure) from METAFONT into TeX. In VTeX this is done by implementing a new `\sincos` primitive command and the `\sine` and `\cosine` dimen registers. Entering `\sincos1pt` fills `\sine` with  $\sin(1^\circ)$  and `\cosine` with  $\cos(1^\circ)$  (notice that one degree is one point). These values can be now used in typesetting

computations. In addition, `\special{R###,###}` is used to tell the drivers about the desired rotation of the font. To avoid re-computing sines/cosines in drivers, we pass their values instead of the angle. Finally, we will need to somewhat modify Hoenig's macros:

```
\catcode`p=12 \catcode`t=12
\gdef\\#1pt{\#1}%
\let\getf=\\
\newdimen\x \newdimen\cos
\newdimen\y \newdimen\sin
\def\initialize{%
  \global\x=0pt\global\y=0pt}

\def\dolist{\afterassignment
  \dodolist\let\next= }
\def\dodolist{\ifx\next\endlist
  \let\next\relax
  \else \\ \let\next\dolist\fi
  \next}
\def\endlist{\endlist}
\def\\{\expandafter\if\space\next
  \addspace\else\point\next\fi}
\newbox\spacebox
\setbox\spacebox=\hbox{ }
\def\addspace{\setbox0=%
  \copy\spacebox\newcoords}
\def\point#1{%
  \setbox0=\hbox{\#1} for \newcoords
  \setbox2=\hbox{\#1} for typesetting
  \wd2=0pt \ht2=0pt \dp2=0pt
  \rlap{\kern\x \raise\y \box2}%
  \newcoords}
\def\newcoords{%
  \global\advance\x by \cos
  \global\advance\y by -\sin}
\def\angletype#1{\initialize
  \leavevmode\setbox1=
  \hbox{\dolist\#1\endlist}\box1}
```

Now, we define

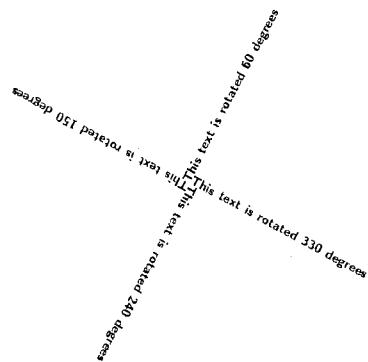
```
\def\tryrotation#1{%
  \setrotation #1pt%
  \def\sinf{%
    \expandafter\getf\the\sine\wd0}%
  \def\cosf{%
    \expandafter\getf\the\cosine\wd0}%
  \special{R\the\cosine,\the\sine}%
  \angletype{%
    This text is rotated #1 degrees}%
  \special{R0,0} % Turn off rotation.
```

and type

```
\vskip-1cm \hskip8cm
\font\anglefont=mvssbx10 \anglefont
\tryrotation{60}%
\tryrotation{150}%
\tryrotation{240}%
\tryrotation{330}%

```

to obtain



Other examples shown in Hoenig's article can be handled similarly.

**Internals.** The changes needed in the *TEX* program are as follows: define new dimension parameters *\sine* and *\cosine* (new codes are *sine\_code* and *cosine\_code*) and new extension primitive *\sin-cos* (using *compute\_sincos* code) and accordingly modify *init\_prim* and *print\_cmd\_chr*. Procedure *do\_extension* receives new case:

```
compute_sincos:
begin
scan_normal_dimen; {angle*1000}
n_sin_cos(cur_val*16);
n_sin:=n_sin div 4096;
n_cos:=n_cos div 4096;
eq_word_define
(dimen_base+ sine_code,-n_sin);
eq_word_define
(dimen_base+cosine_code, n_cos);
end;
```

where *n\_sin* and *n\_cos* are temporary integers (in the METAFONT source, these were macros). Finally, transplant *n\_sin\_cos* as well as the procedures it needs (*pyth\_add*, *make\_fraction*, and *take\_fraction*) from the METAFONT into the *TEX* source. This modification adds about 2K to the *TEX* program.

## Case Study II: Bitmap Graphics Inclusion

**The problem.** A casual study shows that about 10% of articles published in TUGboat deal with

graphics inclusion problems. This should not be unexpected since *TEX*'s design completely ignores the existence of graphics. Graphics inclusion is normally done in one of two ways: either *TEX* allocates space for a graphics box, sets the reference point and passes the name of the graphics file via a *\special*, or graphics are converted into *.tfm/.pk* pairs and *TEX* treats them as characters. The advantage here is that off-the-shelf drivers can be made to print graphics; the disadvantage is the extra conversion pass and, frequently, the need to maintain two copies of the graphics file: in the initial and in the *pk* format. Further problems arise because of the *TEX*'s limit on the number of fonts. Finally, the *.tfm/.pk* approach is not applicable to vector graphics formats. The *\special* approach requires a way to measure the dimensions of the graphics images; it also assumes that the drivers can read (and scale) graphics in several graphics formats (PCX and TIF to start with).

A possible interface for *TEX* follows:

```
\newdimen\graphX \newdimen\graphY
\newbox\gbox % graphics box.
\def\scalegraph#1#2{%
\graphX=1in \divide\graphX by #1
\multiply\graphX by \graphx
\graphY=1in \divide\graphY by #2
\multiply\graphY by \graphy}

\def\makepicbox#1#2#3{%
\sizegraph #3
\scalegraph{#1}{#2}
\setbox\gbox
\hbox{\special{G,#1,#2,#3}}
\ht\gbox=\graphY
\wd\gbox=\graphX}

%Example:
% \makepicbox{300}{300}{test.pic}
```

where *\graphx* and *\graphy* hold the pixel dimensions, set by *\sizegraph*; *\special{G...}* communicates the name of the graphics file to the drivers. The parameters to *\makepicbox* are the "natural" x- and y-resolutions of the picture; if they match the resolutions of the device driver, no scaling is needed.

From the *TEX*'s point of view, the only interesting question is the implementation of *\sizegraph*. There are several ways:

- (1) Hardwire the dimensions inside the *TEX* source; i.e.,

```
\def\scalegraph#1{\graphx640\graphy350}
```

- (2) Read the dimensions from an `\input` file. If the graphics are stored in `graph.pic`, we assume that there is a header file `graph.tex` containing `\graphx640\graphy350`. `\scalegraph`, therefore, will change the file extension to `.tex` and `\input` the file. The problem with this solution is the need for the user to create and maintain the header files. As a minimum, one would require an auxiliary utility for determining the dimensions of graphics (call it `SIZEGRAPH`) and a `MAKE` program for ensuring that all header files are up-to-date.
- (3) Implement `\sizegraph` as an extension primitive; make `\graphx` and `\graphy` dimen registers. This is the original approach used by VTeX. On the positive side, it eliminates the need for header files; on the negative, it burdens the TeX program with the need to know different graphics formats. Another hidden advantage over (2) is that TeX accumulates names of the `\input` files in its string pool; thus in (2) the string pool is likely to overflow on documents that include hundreds of pictures.
- (4) A combination of (2) and (3). Keep a stand-alone `SIZEGRAPH` program and make TeX invoke it whenever it needs to get the dimensions of a graphics image. This appears to be the overall best solution, since `SIZEGRAPH` can now be independently maintained and the extension to TeX is both very small and very general.

### The `\exec` Extension

VTeX extends TeX by adding the `\exec` primitive. `\exec` is implemented as a `message` command with code 2 (code 1 is `\message` and code 2 is `\errmessage`). `\exec` takes two arguments: the external program name and the argument string. Whenever VTeX encounters `\exec`, it stops TeXing and invokes the external program; it resumes the execution once it retains the control. The return code of the external program is reported in the `\errno` integer register. `\exec` allows the following implementation of `\sizegraph`:

```
\def\sizegraph#1{%
  \exec{sizegraph.exe}{#1 > temp.tex}%
  \if\errno0\input temp.tex\else ???\fi
}
```

While `\exec` provides possibly the best way for passing the graphics dimensions to TeX, it can also be used, for instance, to implement `\sin cos` outside

of TeX. Font substitution extensions described below can also be done by `\exec`'ting lookups into auxiliary tables. In fact, the `\exec` command is the ultimate extension: most other extensions discussed in this paper can be implemented through `\exec`; at the same time `\exec` does not seriously infringe on TeX syntax. As will be seen below, `\exec` can even be implemented without any modifications to TeX whatsoever.

The discussion will not be complete without mentioning the `\command` variant of `\exec`. Under MS-DOS, `\command` passes the command string to the command processor, rather than executing the program directly. Thus, `\command` can be used to execute internal commands.

```
\def\command#1{\exec
  {command.com}{/C #1}}
```

### Case Study III: Automatic Index Generation

Another logical application of `\exec` would be an automated index for TeX. The index macros defined in the Appendix E of *The TeXbook* and actually used in formatting the *Computers & Typesetting* series provide excellent tools for generating indices. Unfortunately, these tools cannot be fully used from inside TeX since TeX lacks sorting abilities. Adding `sort` to TeX is an extension that the author would hardly advocate; VTeX's index is constructed by running an auxiliary `IDXSRT` program via `\exec` and then merging the results into the document (`IDXSRT` is capable of sorting and formatting indices in many different ways; in particular, it can remove multiple references to the same item that appears on one page.). The index is constructed by first using the `\icopy` and `\iput` macros, where `\iput` writes the argument into the index file, together with page and/or section number; `\icopy` is simply

```
\def\icopy#1{\#1\iput{\#1}}
```

When it is time to insert the index, we use `\mergeindex`:

```
\def\mergeindex{%
  \immediate\closeout\@indexfile%
  \command{idxsrt \indexparams\%
    \@indexname eraseme.tex}%
  \input eraseme.tex
  \command{erase eraseme.tex}}%
```

where `\indexparams` define the switches to be passed to `IDXSRT`.

## Case Study IV: Font Substitution

In preparing a document, one often needs to change the size (or the attributes) of the font, *regardless of the font used*: it may be desirable to typeset footnotes at eight points and titles at fourteen, regardless of what font changes may appear in the document. For instance, in preparation of this article, the author was hoping to enter

```
\head *The {\tt\char92exec} Extension*
```

However, the `\head` macro scaled the roman font to 12 points and left teletype at 10 points (see previous page). The problem is unresolved in PLAIN  $\text{\TeX}$ ;  $\text{\LaTeX}$  2.09 solves it by providing 800-line long table of font substitutions (`LFONTS.TEX`) plus repeated definitions of `\large`, `\huge`, etc., all over the style files.  $\text{\LaTeX}$ 's solution is only partial: it does not support point sizes not explicitly listed in `LFONTS.TEX`; neither do  $\text{\LaTeX}$ 's tables support non-cm fonts.

Most  $\text{\TeX}$  users would find it greatly desirable to have compact and portable definitions of `\large`, `\small`, etc., that will support all  $\text{\TeX}$  fonts. Since the need to support all possible fonts *precludes* usage of  $\text{\LaTeX}$ -style tables, the effect will be achieved by extending  $\text{\TeX}$ . We add new integer register `\fontscale`. All the `setfont` commands are processed *relatively* to `\fontscale`. For example, if `\fontscale` is set to 1200, `\tt` will invoke teletype at 12, not at 10 points. We can now define

```
\def\huge{\fontscale=2400\setfonts}
\def\large{\fontscale=1200\setfonts}
\def\small{\fontscale=800\setfonts}
\def\tiny{\fontscale=514\setfonts}
\def\outl{\outline=1\setfonts}
\def\fillp#1{\fillpattern=#1\setfonts}
\def\setfonts{\the\font\setmathfonts}
\def\setmathfonts{%
  \textfont0\textfont0
  \textfont1\textfont1
  ...
  ...
  ...
  ...
  \scriptscriptfont15\scriptscriptfont15}
```

and so on. Notice that after setting `\fontscale` we need to reissue the last font command (`\the\font`) to ensure that the current font changes.

**Remark:** The drawback of this definition of `\setfonts` above is the loading of math fonts caused by each font change switch regardless of whether math fonts will be needed. An alternative is to declare

```
\newif\ifnewmath
\def\setfonts{\the\font\newmathtrue}
\everymath{\ifnewmath\setmathfonts}
```

```
\newmathfalse\fi}
```

which will eliminate unnecessary font loads but may or may not conflict with other usage of `\everymath`.

**Internals.** `\fontscale` (and its companion `\bold`, `\smallcaps`, `\shadow`, `\outline`, `\fillpattern`, `\slant`, and `\aspect`) are simply additional integer parameters. As mentioned above, these are added by modifying the `init_prim` and `print_cmd_chr` routines. The standard values (set by `Init\TeX`) are 1000 for `\fontscale` and `\aspect` and 0 for others.

The tricky part is the modification of the `prefixed_command` routine that handles font assignments. We start by replacing the standard

```
set_font:
  define(cur_font_loc,data,cur_chr);
```

with

```
set_font:
  define(cur_font_loc,data,
    fsubst(cur_chr));
```

The `fsubst` procedure returns with unmodified `cur_chr` if one of three events holds:

- 1) the program is run in  $\text{\TeX}$ -compatibility mode, where fonts cannot be substituted;
- 2) its argument is the `nullfont` (`cur_chr=0`); or
- 3) all eight relevant integer registers (`\fontscale` through `\aspect`) hold default values.

If none of the above is true, `fsubst` retrieves the parameters for the font-in-question, multiplies the magnifications and the aspect ratios, adds the slants, and applies the exclusive-or to the remaining parameters. It next verifies that the font with required parameters has not yet been loaded and calls `read_font_info` to create it. Finally, it returns the font number obtained from `read_font_info`.

A similar change in the `def_family` subcase of the `prefixed_command` routine makes the `\textfont`, `\scriptfont`, and `\scriptscriptfont` relative.

**Invisible fonts/color separation.** An additional benefit is the ability to implement color separation via invisible fonts. Assuming that the `\fillpattern0` is 100 that the `\fillpattern1` is 0

```
\def\print{\fillp{0}}
\def\dontprint{\fillpattern{1}}
\def\redcopy{
  \def\red{\print}
  \def\green{\dontprint}
  \def\blue{\bluecopy}
\def\bluecopy{
  \def\red{\dontprint}
```

```
\def\green{\dontprint}
\def\blue{\tprint}
```

to allow selective printing of color planes.

**Math rules.** This pattern and color selection scheme needs a modification to be useful in math mode, where symbols are often built from both characters and *rules*. As given above, the `\fillp` command affects only the character part, creating misfits like



instead of



Currently, VTeX solves the problem by defining

```
\def\fillp#1{\fillpattern=#1\setfonts
\special{F#1}}
```

where `\special{F#1}` instructs the device drivers to start shading rules.

Yet another difficulty is the possibility of shading that spans from one page to another. Unless the `\special` is re-issued on each page, a device driver would not see it if it processes the second page before the first. Solutions with different degrees of generality are possible.

### A Special Note to a TeX Purist

Most of the extensions described in this paper can be used *without any changes* to TeX program. For instance, to use `\exec`, without implementing it we will write a loader program that traps screen and keyboard I/O and loads TeX, waiting for infamous

```
! Undefined control sequence.
<*> \exec
      {wipetext}{*.log}
?
```

(make sure that `\exec` and its arguments are on a line by itself, so they will be echoed on the next line.) The loader now swaps TeX out of memory, performs the `\exec`, swaps TeX in, and inserts d8 into the TeX's mouth to delete now-unneeded tokens. While the author found this solution lacking in performance, it has been tested and worked with PC implementations of TeX.

### A Late Note

After this paper has been presented at the TUG conference, a couple of participants noticed yet another usage for the `\exec`: as a security-breaching vehicle. Indeed, it is possible to write a TeX program to write, for example, a C program, and then `\exec` to compile, link and run it.

### Conclusions

The author hopes that this paper will be helpful in encouraging further development of TeX.

July 27 to 30, 1992

# 13th Annual T<sub>E</sub>X Users Group Meeting



# PORTLAND, OREGON

▲ Mark your calendars and join us in Portland, the home of 20-pound salmon and 20-story buildings. Ride light rail trains over cobblestone streets, ski Mt. Hood and attend the symphony in the same day—even in July. A friendly city, Portland charms its visitors with a variety of attractions including:

Windsurfing

A trip up the Columbia River on a sternwheeler

Tours of the wine region

The Metro Washington Park Zoo

Portland Center for the Performing Arts

Oaks Amusement Park

Oregon Art Institute

Scenic Washington County

Oregon Museum of Science and Industry

World Forestry Center

Mt. Hood

Portland Saturday Market for arts and crafts

Of special interest to TUG Meeting attendees may be the 11th Annual Mt. Hood Festival of Jazz to be held August 1st and 2nd in Gresham, Oregon, a suburb of Portland.

For a complete visitors' guide, *The Portland Book*, call the Portland Visitors' Center at (800) 345-3214.

## T<sub>E</sub>X in Context

### Resources, Support Tools, and Comparative Studies

▲ During four information-packed days, we'll delve into front-ends for T<sub>E</sub>X, inclusion of graphics within T<sub>E</sub>X documents as well as exportation of T<sub>E</sub>X output to other graphics programs, comparisons of implementations of T<sub>E</sub>X on microcomputers, network access and resources, educational issues, and translation between T<sub>E</sub>X and word-processors.

Presentations  
Workshops  
Networking Luncheons  
Exhibits  
Panel discussions  
Classes

▲ We'll meet and stay at the Benson Hotel, Portland's premier hotel recently restored to its grand stature of the early 1900s. A registered historic landmark, the Benson was built by Oregon lumberman, Simon Benson using elaborate craftsmanship and imported wood interiors. Special TUG rates: \$89/night (available until June 26 only.)

▲ Program coordinator:

Mimi Lafrenz  
*ETP Services Co.*

Program committee:

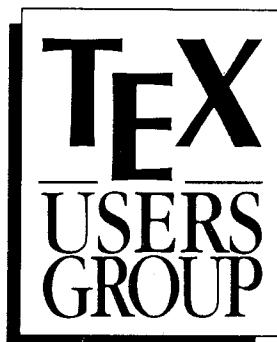
Helen Gibson  
*Wellcome Institute*

Doug Henderson  
*Blue Sky Research*

Ron Whitney  
*T<sub>E</sub>X Users Group*

▲ Watch your mail and future issues of *TUGboat* and *T<sub>E</sub>X & TUG News* for more details. In the meantime, if you have questions, contact:

**T<sub>E</sub>X Users Group**  
Phone: (401) 751-7760  
Fax: (401) 751-1071  
e-mail: [tug@math.ams.com](mailto:tug@math.ams.com)  
P.O. Box 9506  
Providence, RI 02940



## Institutional Members

The Aerospace Corporation,  
*El Segundo, California*

Air Force Institute of Technology,  
*Wright-Patterson AFB, Ohio*

American Mathematical Society,  
*Providence, Rhode Island*

ArborText, Inc.,  
*Ann Arbor, Michigan*

ASCII Corporation,  
*Tokyo, Japan*

Belgrade University,  
Faculty of Mathematics,  
*Belgrade, Yugoslavia*

Brookhaven National Laboratory,  
*Upton, New York*

CERN, *Geneva, Switzerland*

Brown University,  
*Providence, Rhode Island*

California Institute of Technology,  
*Pasadena, California*

Calvin College,  
*Grand Rapids, Michigan*

Carleton University,  
*Ottawa, Ontario, Canada*

Centre Inter-Régional de  
Calcul Électronique, CNRS,  
*Orsay, France*

College of William & Mary,  
Department of Computer Science,  
*Williamsburg, Virginia*

Communications  
Security Establishment,  
Department of National Defence,  
*Ottawa, Ontario, Canada*

Construcciones Aeronauticas, S.A.,  
CAE-Division de Proyectos,  
*Madrid, Spain*

DECUS, Electronic Publishing  
Special Interest Group,  
*Marlboro, Massachusetts*

Department of National Defence,  
*Ottawa, Ontario, Canada*

E. S. Ingenieros Industriales,  
*Sevilla, Spain*

Edinboro University  
of Pennsylvania;  
*Edinboro, Pennsylvania*

Elsevier Science Publishers B.V.,  
*Amsterdam, The Netherlands*

European Southern Observatory,  
*Garching bei München, Federal Republic of Germany*

Fermi National Accelerator  
Laboratory, *Batavia, Illinois*

Florida State University,  
Supercomputer Computations  
Research, *Tallahassee, Florida*

Fordham University,  
*Bronx, New York*

General Motors  
Research Laboratories,  
*Warren, Michigan*

Grinnell College,  
Computer Services,  
*Grinnell, Iowa*

GTE Laboratories,  
*Waltham, Massachusetts*

Hatfield Polytechnic,  
Computer Centre,  
*Herts, England*

Hughes Aircraft Company,  
Space Communications Division,  
*Los Angeles, California*

Hungarian Academy of Sciences,  
Computer and Automation  
Institute, *Budapest, Hungary*

IBM Corporation,  
Scientific Center,  
*Palo Alto, California*

Institute for Advanced Study,  
*Princeton, New Jersey*

Institute for Defense Analyses,  
Communications Research  
Division, *Princeton, New Jersey*

Intevep S. A., *Caracas, Venezuela*

Iowa State University,  
*Ames, Iowa*

The Library of Congress,  
*Washington D.C.*

Los Alamos National Laboratory,  
University of California,  
*Los Alamos, New Mexico*

Louisiana State University,  
*Baton Rouge, Louisiana*

MacroSoft, *Warsaw, Poland*

Marquette University,  
Department of Mathematics,  
Statistics and Computer Science,  
*Milwaukee, Wisconsin*

Masaryk University,  
*Brno, Czechoslovakia*

Mathematical Reviews,  
American Mathematical Society,  
*Ann Arbor, Michigan*

Max Planck Institut  
für Mathematik,  
*Bonn, Federal Republic of Germany*

McGill University,  
*Montréal, Québec, Canada*

Michigan State University,  
Mathematics Department,  
*East Lansing, Michigan*

NASA Goddard  
Space Flight Center,  
*Greenbelt, Maryland*

National Institutes of Health,  
*Bethesda, Maryland*

National Research Council  
Canada, Computation Centre,  
*Ottawa, Ontario, Canada*

Naval Postgraduate School,  
*Monterey, California*

New York University,  
Academic Computing Facility,  
*New York, New York*

Nippon Telegraph &  
Telephone Corporation,  
Software Laboratories,  
*Tokyo, Japan*

Northrop Corporation,  
*Palos Verdes, California*

The Open University,  
Academic Computing Services,  
*Milton Keynes, England*

Pennsylvania State University,  
Computation Center,  
*University Park, Pennsylvania*

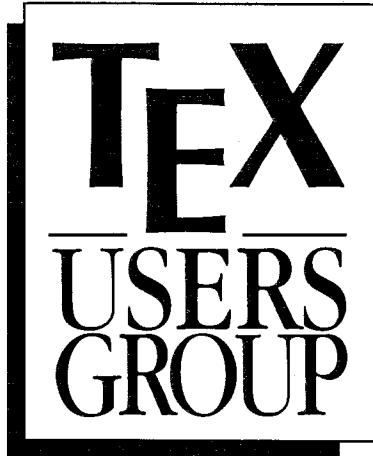
Personal TeX, Incorporated,  
*Mill Valley, California*

Princeton University,  
*Princeton, New Jersey*

Purdue University,  
*West Lafayette, Indiana*

Queens College,  
*Flushing, New York*

- Rice University,  
Department of Computer Science,  
*Houston, Texas*
- Roanoke College,  
*Salem, VA*
- Rogaland University,  
*Stavanger, Norway*
- Ruhr Universität Bochum,  
Rechenzentrum,  
*Bochum, Federal Republic of Germany*
- Rutgers University, Hill Center,  
*Piscataway, New Jersey*
- St. Albans School,  
*Mount St. Alban, Washington, D.C.*
- Sandia National Laboratories,  
*Albuquerque, New Mexico*
- Smithsonian Astrophysical Observatory, Computation Facility,  
*Cambridge, Massachusetts*
- Software Research Associates,  
*Tokyo, Japan*
- Space Telescope Science Institute,  
*Baltimore, Maryland*
- Springer-Verlag,  
*Heidelberg, Federal Republic of Germany*
- Springer-Verlag New York, Inc.,  
*New York, New York*
- Stanford Linear Accelerator Center (SLAC),  
*Stanford, California*
- Stanford University,  
Computer Science Department,  
*Stanford, California*
- Talaris Systems, Inc.,  
*San Diego, California*
- Texas A & M University,  
Department of Computer Science,  
*College Station, Texas*
- UNI-C, *Aarhus, Denmark*
- United States Military Academy,  
*West Point, New York*
- University of Alabama,  
*Tuscaloosa, Alabama*
- University of British Columbia,  
Computing Centre,  
*Vancouver, British Columbia, Canada*
- University of British Columbia,  
Mathematics Department,  
*Vancouver, British Columbia, Canada*
- University of Calgary,  
*Calgary, Alberta, Canada*
- University of California, Berkeley,  
Space Astrophysics Group,  
*Berkeley, California*
- University of California, Irvine,  
Information & Computer Science,  
*Irvine, California*
- University of California, Los Angeles, Computer Science Department Archives,  
*Los Angeles, California*
- University of Canterbury,  
*Christchurch, New Zealand*
- Universidade de Coimbra,  
*Coimbra, Portugal*
- University College,  
*Cork, Ireland*
- University of Crete,  
Institute of Computer Science,  
*Heraklio, Crete, Greece*
- University of Delaware,  
*Newark, Delaware*
- University of Exeter,  
Computer Unit,  
*Exeter, Devon, England*
- University of Glasgow,  
Department of Computing Science,  
*Glasgow, Scotland*
- University of Groningen,  
*Groningen, The Netherlands*
- University of Heidelberg,  
Computing Center Heidelberg,  
*Germany*
- University of Illinois at Chicago,  
Computer Center,  
*Chicago, Illinois*
- University of Kansas,  
Academic Computing Services,  
*Lawrence, Kansas*
- Universität Koblenz-Landau,  
*Koblenz, Federal Republic of Germany*
- University of Maryland,  
Department of Computer Science,  
*College Park, Maryland*
- University of Maryland  
at College Park,  
Computer Science Center,  
*College Park, Maryland*
- University of Massachusetts,  
*Amherst, Massachusetts*
- University of Oslo,  
Institute of Informatics,  
*Blindern, Oslo, Norway*
- University of Oslo,  
Institute of Mathematics,  
*Blindern, Oslo, Norway*
- University of Ottawa,  
*Ottawa, Ontario, Canada*
- University of Salford,  
*Salford, England*
- University of Southern California,  
Information Sciences Institute,  
*Marina del Rey, California*
- University of Stockholm,  
Department of Mathematics,  
*Stockholm, Sweden*
- University of Texas at Austin,  
*Austin, Texas*
- University of Washington,  
Department of Computer Science,  
*Seattle, Washington*
- University of Western Australia,  
Regional Computing Centre,  
*Nedlands, Australia*
- Uppsala University,  
*Uppsala, Sweden*
- Vereinigte Aluminium-Werke AG,  
*Bonn, Federal Republic of Germany*
- Villanova University,  
*Villanova, Pennsylvania*
- Vrije Universiteit,  
*Amsterdam, The Netherlands*
- Washington State University,  
*Pullman, Washington*
- Widener University,  
Computing Services,  
*Chester, Pennsylvania*
- Worcester Polytechnic Institute,  
*Worcester, Massachusetts*
- Yale University,  
Department of Computer Science,  
*New Haven, Connecticut*



**Complete and return this form with payment to:**

TEX Users Group  
Membership Department  
P. O. Box 594  
Providence, RI 02901 USA  
Telephone: (401) 751-7760  
FAX: (401) 751-1071  
Email: [tug@Math.AMS.com](mailto:tug@Math.AMS.com)

**Membership is effective** from January 1 to December 31 and includes subscriptions to *TUGboat*, *The Communications of the T<sub>E</sub>X Users Group* and the TUG newsletter, *T<sub>E</sub>X and TUG News*. Members who join after January 1 will receive all issues published that calendar year.

---

**For more information ...**

Whether or not you join TUG now, feel free to return this form to request more information. Be sure to include your name and address in the spaces provided to the right.

**Check all items you wish to receive below:**

- Institutional membership information
- Course and meeting information
- Advertising rates
- Products/publications catalogue
- Public domain software catalogue
- More information on T<sub>E</sub>X

## Individual Membership Application

Name \_\_\_\_\_

Institutional affiliation, if any \_\_\_\_\_

Position \_\_\_\_\_

Address (business or home (circle one)) \_\_\_\_\_  
\_\_\_\_\_

City \_\_\_\_\_

State or Country \_\_\_\_\_ Zip \_\_\_\_\_

Daytime telephone \_\_\_\_\_ FAX \_\_\_\_\_

Email addresses (*please specify networks, as well*) \_\_\_\_\_  
\_\_\_\_\_

I am also a member of the following other T<sub>E</sub>X organizations:

Specific applications or reasons for interest in T<sub>E</sub>X:

Hardware on which T<sub>E</sub>X is used:

*Computer and operating system*                           *Output device/printer*

\_\_\_\_\_

There are two types of TUG members: regular members, who pay annual dues of \$60; and full-time student members, whose annual dues are \$50. Students must include verification of student status with their applications.

Please indicate the type of membership for which you are applying:

Regular @ \$60     Full-time student @ \$50

Amount enclosed for 1992 membership:                            \$ \_\_\_\_\_

*(Prepayment in US dollars drawn on a US bank is required)*

Check/money order payable to T<sub>E</sub>X Users Group enclosed  
 Charge to MasterCard/VISA

Card # \_\_\_\_\_ Exp. date \_\_\_\_\_

Signature \_\_\_\_\_



Complete and return this form  
with payment to:

TEX Users Group  
Membership Department  
P.O. Box 594  
Providence, RI 02901 USA

**Bank transfers**

TEX Users Group, #002-031375  
Hospital Trust National Bank  
One Hospital Trust Plaza  
Providence, RI 02903  
USA

**Membership is effective** from January 1 to December 31. Members who join after January 1 will receive all issues of *TUGboat* published that calendar year.

**For more information ...**

**Correspondence**

TEX Users Group  
653 North Main Street  
P.O. Box 9506  
Providence, RI 02940  
USA  
  
Telephone: (401) 751-7760  
Fax: (401) 751-1071  
Email: [tug@math.ams.com](mailto:tug@math.ams.com)

Whether or not you join TUG now,  
feel free to return this form to  
request more information.

**Check all items you wish to receive below:**

- Course and meeting information
- Products/publications catalogue
- Public domain software catalogue

## Institutional Membership Application

Institution or Organization \_\_\_\_\_

Principal contact \_\_\_\_\_

Address \_\_\_\_\_

City \_\_\_\_\_

State or Country \_\_\_\_\_ Zip \_\_\_\_\_

Daytime telephone \_\_\_\_\_ FAX \_\_\_\_\_

Email addresses (*please specify networks, as well*) \_\_\_\_\_

Each Institutional Member is entitled to:

- designate a number of individuals to have full status as TUG individual members;
- take advantage of reduced rates for TUG meetings and courses for *all* staff members;
- be acknowledged in every issue of *TUGboat* published during the membership year.

Educational institutions receive a \$100 discount in the membership fee. The three basic categories of Institutional Membership each include a certain number of individual memberships. Additional individual memberships may be obtained at the rates indicated. Fees are as follows:

Category	Rate (educ./non-educ.)	Add'l mem.
A (includes 7 memberships)	\$ 540 / \$ 640	\$50 ea.
B (includes 12 memberships)	\$ 815 / \$ 915	\$50 ea.
C (includes 30 memberships)	\$1710 / \$1810	\$40 ea.

Please indicate the type of membership for which you are applying:

Category \_\_\_\_\_ + \_\_\_\_\_ additional individual memberships

Amount enclosed for 1992 membership: \$ \_\_\_\_\_

- Check/money order payable to TEX Users Group enclosed  
(*payment is required in US dollars drawn on a US bank*)
- Bank transfer bank \_\_\_\_\_  
ref # \_\_\_\_\_
- Charge to MasterCard/VISA

Card # \_\_\_\_\_ Exp. date \_\_\_\_\_

Signature \_\_\_\_\_

Please attach a corresponding list of individuals whom you wish to designate as TUG individual members. Minimally, we require names and addresses so that TUG publications may be sent directly to these individuals, but we would also appreciate receiving the supplemental information regarding phone numbers, email addresses, TEX interests, and hardware configurations as requested on the TUG Individual Membership Application form. For this purpose, the latter application form may be photocopied and mailed with this form.

## TeX Consulting and Production Services

### North America

**AMERICAN MATHEMATICAL SOCIETY**  
 P. O. Box 6248, Providence, RI 02940; (401) 455-4060  
 Typesetting from DVI files on an Autologic APS Micro-5  
 or an Agfa Compugraphic 9600 (PostScript).  
 Times Roman and Computer Modern fonts.  
 Composition services for mathematical and technical books  
 and journal production.

**ANAGNOSTOPOULOS, Paul C.**  
 433 Rutland Street, Carlisle, MA 01741; (508) 371-2316  
 Composition and typesetting of high-quality books and  
 technical documents. Production using Computer Modern  
 or any available PostScript fonts. Assistance with book  
 design. I am a computer consultant with a Computer  
 Science education.

**ARBORTEXT, Inc.**  
 535 W. William, Suite 300, Ann Arbor, MI 48103;  
 (313) 996-3566  
 Typesetting from DVI files on an Autologic APS-5.  
 Computer Modern and standard Autologic fonts.  
 TeX installation and applications support.  
 TeX-related software products.

**ARCHETYPE PUBLISHING, Inc.,**  
 Lori McWilliam Pickert  
 P. O. Box 6567, Champaign, IL 61821; (217) 359-8178  
 Experienced in producing and editing technical journals  
 with TeX; complete book production from manuscript to  
 camera-ready copy; TeX macro writing including complete  
 macro packages; consulting.

**THE BARTLETT PRESS, Inc.,**  
 Frederick H. Bartlett  
 Harrison Towers, 6F, 575 Easton Avenue,  
 Somerset, NJ 08873; (201) 745-9412  
 Vast experience: 100+ macro packages, over 30,000 pages  
 published with our macros; over a decade's experience in all  
 facets of publishing, both TeX and non-TeX; all services  
 from copyediting and design to final mechanicals.

**COWAN, Dr. Ray F.**  
 141 Del Medio Ave. #134, Mountain View, CA 94040;  
 (415) 949-4911  
*Ten Years of TeX and Related Software Consulting  
 Books, Documentation, Journals, and Newsletters*  
 TeX & LATEX macropackages, graphics; PostScript language  
 applications; device drivers; fonts; systems.

**DOWNES, Michael**  
 49 Weeks Street, North Smithfield, RI 02895;  
 (401) 762-3715  
 Instruction in AMS-TeX, AMS-LATEX, plain TeX, and  
 advanced macro writing. Custom documentstyles.  
 Consulting: ■ advanced mathematical typesetting topics;  
 ■ tuning mathematics fonts; ■ getting the most out of TeX  
 in a production environment. Troubleshooting.

**ELECTRONIC TECHNICAL PUBLISHING  
 SERVICES CO.**  
 2906 Northeast Glisan Street, Portland, Oregon 97232-3295;  
 (503) 234-5522; FAX: (503) 234-5604  
 Total concept services include editorial, design, illustration,  
 project management, composition and prepress. Our years

of experience with TeX and other electronic tools have  
 brought us the expertise to work effectively with publishers,  
 editors, and authors. ETP supports the efforts of the TeX  
 Users Group and the world-wide TeX community in the  
 advancement of superior technical communications.

**HOENIG, Alan**  
 17 Bay Avenue, Huntington, NY 11743; (516) 385-0736  
 TeX typesetting services including complete book  
 production; macro writing; individual and group  
 TeX instruction.

**KUMAR, Romesh**  
 1549 Ceals Court, Naperville, IL 60565; (708) 972-4342  
 Beginners and intermediate group/individual instruction  
 in TeX. Development of TeX macros for specific purposes.  
 Using TeX with FORTRAN for custom-tailored software.  
 Flexible hours, including evenings and weekends.

**MAGUS, Kevin W. Thompson**  
 P. O. Box 390965, Mountain View CA 94039-0965;  
 (800) 848-8037; (415) 940-1109; magus@cup.portal.com  
 LATEX consulting from start to finish. Layout design  
 and implementation, macro writing, training, phone  
 support, and publishing. Can take LATEX files and return  
 camera ready copy. Knowledgeable about long document  
 preparation and mathematical formatting.

**OGAWA, Arthur**  
 920 Addison, Palo Alto, CA 94301; (415) 323-9624  
 Experienced in book production, macro packages,  
 programming, and consultation. Complete book production  
 from computer-readable copy to camera-ready copy.

**QUIXOTE, Don Hosek**  
 440F Grinnell, Claremont, CA 91711; (714) 625-0147  
 Complete line of TeX, LATEX, and METAFONT services  
 including custom LATEX style files, complete book  
 production from manuscript to camera-ready copy;  
 custom font and logo design; installation of customized  
 TeX environments; phone consulting service; database  
 applications and more.  
 Call for a free estimate.

**RICHERT, Norman**  
 1614 Loch Lake Drive, El Lago, TX 77586;  
 (713) 326-2583  
 TeX macro consulting.

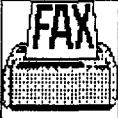
**TEXNOLOGY, Inc., Amy Hendrickson**  
 57 Longwood Ave., Brookline, MA 02146;  
 (617) 738-8029.  
 TeX macro writing (author of MacroTeX); custom macros  
 written to meet publisher's or designer's specifications;  
 instruction.

### Outside North America

**TYPOTEX LTD.**  
 Electronical Publishing, Battyány u. 14. Budapest, Hungary  
 H-1015; (036) 11152 337  
 Editing and typesetting technical journals and books with  
 TeX from manuscript to camera ready copy. Macro writing,  
 font designing, TeX consulting and teaching.

**Index of Advertisers**

- 461, 465 American Mathematical Society  
 467 ArborText  
 Cover 3 Blue Sky Research  
 466 Computer Composition  
 456 Electronic Technical Publishing Services  
 455 Yannis Haralambous  
 454 job ctl  
 457 K-Talk Communications  
 458, 459 Kinch Computer Company  
 454, 460 MicroPress, Inc.  
 465 Micro Programs, Inc.  
 462 Personal T<sub>E</sub>X Inc.  
 464 TCI Software  
 463 Type 2000  
 468 Y&Y



*faxpak* provides Group 3 facsimile capabilities to networked SUNs and XENIX 2.3.2 and compatible systems. Supports SIERRA type modems such as WORLDPORT's 2496 and "Class Two" modems.

File Formats: • Plain ASCII • T<sub>E</sub>X and L<sub>A</sub>T<sub>E</sub>X<sup>1</sup> • POSTSCRIPT text and graphics • SUN raster and other bitmaps • Easily extended to any bitmap or file format.

Configurable Options: • Multiple phone lines • View received faxes on screen • Aliases, distributions lists, batch jobs • Complex permissions scheme or unrestricted access • Departmental or system wide "cover pages" • Page numbering • Pasting up of bitmaps • Restriction of transmissions to "off peak" rates • "Pickup Mode" to avoid resending confirmed pages after errors • Accounting.

*faxpak* comes with a site licence and costs \$360. plus \$25 for shipping by AIR MAIL.. Details from: job ctl, Klaus Schallhorn, 28 Belgrave St., Penzance, Cornwall, TR18 2BJ, UK. FAX +44 736 330083, <faxinfo@cnix.uuxp>.

<sup>1</sup>This ad has been faxed by *faxpak*.

**PXLGen™**

**Finally, you can use superior MicroPress scalable typefaces with your version of T<sub>E</sub>X.**

- Choose from 150+ quality typefaces
- Generate PXL/PK files in seconds
- Font effects: compressed/expanded fonts, shading, outline, smallcaps, and more
- Creates matching TFM files
- As low as \$10 per font

Contact MicroPress for pricing and availability for your CPU.

**MICROPRESS INC.**

68-30 HARROW STREET, FOREST HILLS, NY 11375  
 TEL: 718-575-1816 FAX: 718-575-8038



PXLGen is a trademark of MicroPress Inc. Other products mentioned are trademarks of their respective companies.

كَلَانِعٌ بَلْ هُمْ أَضَلُّ أُولَئِكَ هُمُ الْغَافِلُونَ  
 رِيشِيمَهُ عَلَيْهِ شَرِيْهِ مَفْرِيمَهُ لِمَعَ  
 تَأْسِيْلَهُ لِتَحْصِيلَهُ حَسَدَ  
 بَلْ وَعَزَّزَهُ شَرِيْهِ مَفْرِيمَهُ  
 ~ Ήμος δ' ἡριγένεια φάνη ρόδοδάκτ  
 ΕΓΙΤΟΤΤΩΝΑΓΗΝΕΓΚΑΝΟΙΦΩΚΙΓΤΛΑΙΔΑ  
 ΠΕΤΤΕΔΑΤΗΟΚΑΠΤΑΤΗΑΛΑΤΖΑΝΙΠΗΣ  
 Ac ȝrylce þing zerunðas fop folcer ry  
 Ihr naht euch wieder, schwankende Gestalten,  
 Von der Tunøwe, als si gat und



## SCHOLAR TeX®

Use the combined power of TeX, Metafont and PostScript® to create high quality documents providing classical and modern Arabic, Persian, Ottoman Turkish, Pashto, Urdu, Malay, classical Hebrew, Ivrit, Yiddish, Syriac Estrangelo, Armenian, Greek, epigraphical Greek & Latin, Saxon, old German Fraktur and Schwabacher (*forthcoming*: Glagolitic, old Church Cyrillic, Byzantine Greek, Coptic, old Irish, Syriac Serto and Uiguric Mongolian). • All fonts in pk, EPSF, PostScript® Type 1 and TrueType™ Format • User-defined transcription for input and output of Semitic languages & virtual fonts used for accented Arabic characters • Continuous support for improvements & additions.

*Individuals:* \$200 (specify Macintosh®-Textures™, Macintosh®-OzTex or PC); for the sources (in Metafont, WEB and PostScript®) additional \$100. *Institutions, Publishers:* \$500 (sources included). Orders and information from: Yannis Haralambous, 101/11 rue Breughel, 59650 Villeneuve d'Ascq, France, Fax (33) 20.91.05.64.  
ScholarTeX is a registered trademark of Yannis Haralambous.

# The solution is ETP.

$$\Delta\mathcal{P} = \sum_W \left[ Q_{\text{IPR}} \int_{4-1-87}^{\infty} (D_p + D_m + D_s)^T + \epsilon(P_m - I_P)dt \right]$$

≡ ETP

ETP Services offers solutions to the problems facing the publishers of technical books and journals, with a complete array of composition-related services.

#### **Electronic Technical Publishing Services Company**

2906 N.E. Glisan Street  
Portland, Oregon 97232  
503-234-5522 • FAX: 503-234-5604  
[mimi@etp.com](mailto:mimi@etp.com)

Publishing Companion® translates

# WordPerfect

to

# T<sub>E</sub>X or L<sub>A</sub>T<sub>E</sub>X

## IN ONE EASY STEP!

With Publishing Companion, you can publish documents using T<sub>E</sub>X or L<sub>A</sub>T<sub>E</sub>X with **little or no T<sub>E</sub>X knowledge.** Your WordPerfect files are translated into T<sub>E</sub>X or L<sub>A</sub>T<sub>E</sub>X files, so anyone using this simple word processor can immediately begin typesetting their own documents!

Publishing Companion translates EQUATIONS, FOOTNOTES, ENDNOTES, FONT STYLES, and much more!

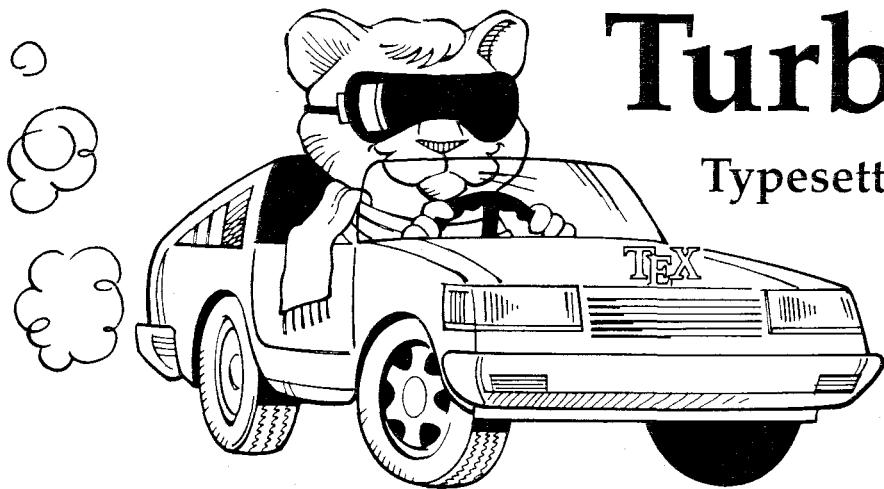
Retail Price .....	\$249.00
Academic Discount Price .....	\$199.00

For more information or to place an order, call or write:



30 West First Ave, Suite 100  
Columbus, Ohio 43201  
(614)294-3535  
FAX (614)294-3704

**TYPESET QUALITY WITH THE EASE OF WORD PROCESSING**



# TurboTeX

Typesetting Software

Executables \$150  
With Source \$300

Now  
Windows  
Compatible!

**N**OW YOU CAN run the TeX typesetting system in the powerful and convenient graphical environment of Microsoft Windows, with the new Windows-compatible TurboTeX Release 3.1.

TurboTeX brings you the latest TeX 3.1 and METAFONT 2.7 standards and certifications: preloaded plain TeX, LATEX, AMS-TeX and AMS-LATEX, METAFONT, preview for EGA/VGA displays, Computer Modern and LATEX fonts, and printer drivers for HP LaserJet and DeskJet, PostScript, and Epson LQ and FX dot-matrix printers. This wealth of software runs on your IBM PC (MS-DOS, Windows, or OS/2), UNIX, or VAX/VMS system.

■ **Best-selling Value:** TurboTeX sets the standard for power and value among TeX implementations: one price buys a complete, commercially-hardened typesetting system. *Computer* magazine recommended it as "the version of TeX to have," *IEEE Software* called it "industrial strength," and thousands of satisfied users worldwide agree.

TurboTeX gets you started quickly, installing itself automatically under MS-DOS or Microsoft Windows, and compiling itself automatically under UNIX. The 90-page User's Guide includes generous examples and a full index, and leads you step-by-step through installing and using TeX and METAFONT.

■ **Classic TeX for Windows.** Even if you have never used Windows on your PC, the speed and power of TurboTeX will convince you of the benefits. While the TeX command-

line options and TeXbook interaction work the same, you also can control TeX using friendly icons, menus, and dialog boxes. Windows protected mode frees you from MS-DOS limitations like DOS extenders, overlay swapping, and scarce memory. You can run long TeX formatting or printing jobs in the background while using other programs in the foreground.

■ **MS-DOS Power, Too:** TurboTeX still includes the plain MS-DOS programs. Even without expanded memory hardware, our virtual memory simulation provides the same sized TeX that runs on multi-megabyte mainframes, with capacity for large documents, complicated formats, and demanding macro packages.

■ **Source Code:** The portable C source to TurboTeX consists of over 100,000 lines of generously commented TeX, TurboTeX, METAFONT, previewer, and printer driver source code, including: our WEB system in C; PASCHAL, our proprietary Pascal-to-C translator; Windows menus and text-mode interface library; and preloading, virtual memory, and graphics code, all meeting C portability standards like ANSI and K&R.

■ **Availability & Requirements:** TurboTeX executables for IBM PC's include the User's Guide and require 640K, hard disk, and MS-DOS 3.0 or later. Windows extensions require Microsoft Windows 3.0. Order source code (includes Programmer's Guide) for other machines. On the PC, source compiles with Microsoft C 5.0 or later (and Windows SDK for Windows extensions), Watcom C 8.0, or Borland C++ 2.0; other op-

erating systems need a 32-bit C compiler supporting UNIX standard I/O. Media is 360K 5-1/4" or 720K 3-1/2" PC floppy disks (please specify).

■ **Upgrade at Low Cost.** If you have TurboTeX Release 3.0, upgrade to the latest version for just \$40 (executables) or \$80 (including source). Or, get either applicable upgrade free when you buy the AP-TeX fonts (see facing page) for \$200!

■ **No-risk trial offer:** Examine the documentation and run the PC TurboTeX for 10 days. If you are not satisfied, return it for a 100% refund or credit. (Offer applies to PC executables only.)

■ **Free Buyer's Guide:** Ask for the free, 70-page Buyer's Guide for details on TurboTeX and dozens of TeX-related products: previewers, TeX-to-FAX and TeX-to-Ventura/Pagemaker translators, optional fonts, graphics editors, public domain TeX accessory software, books and reports.

## Ordering TurboTeX

Ordering TurboTeX is easy and delivery is fast, by phone, FAX, or mail. Terms: Check with order (free media and ground shipping in US), VISA, Mastercard (free media, shipping extra); Net 30 to well-rated firms and public agencies (shipping and media extra). Discounts available for quantities or resale. International orders gladly expedited via Air or Express Mail.

### The Kinch Computer Company

PUBLISHERS OF TURBOTeX  
501 South Meadow Street  
Ithaca, New York 14850 USA  
Telephone (607) 273-0222  
FAX (607) 273-0484

# AP-T<sub>E</sub>X Fonts

## T<sub>E</sub>X-compatible Bit-Mapped Fonts Identical to Adobe PostScript Typefaces

If you are hungry for new T<sub>E</sub>X fonts, here is a feast guaranteed to satisfy the biggest appetite! The AP-T<sub>E</sub>X fonts serve you a banquet of gourmet delights: 438 fonts covering 18 sizes of 35 styles, at a total price of \$200. The AP-T<sub>E</sub>X fonts consist of PK and TFM files which are exact T<sub>E</sub>X-compatible equivalents (including "hinted" pixels) to the popular PostScript name-brand fonts shown at the right. Since they are directly compatible with any standard T<sub>E</sub>X implementation (including kerning and ligatures), you don't have to be a T<sub>E</sub>X expert to install or use them.

When ordering, specify resolution of 300 dpi (for laser printers), 180 dpi (for 24-pin dot matrix printers), or 118 dpi (for previewers). Each set is on ten 360 KB 5-1/4" PC floppy disks. The \$200 price applies to the first set you order; order additional sets at other resolutions for \$60 each. A 30-page user's guide fully explains how to install and use the fonts. Sizes included are 5, 6, 7, 8, 9, 10, 11, 12, 14.4, 17.3, 20.7, and 24.9 points; headline styles (equivalent to Times Roman, Helvetica, and Palatino, all in bold) also include sizes 29.9, 35.8, 43.0, 51.6, 61.9, and 74.3 points.

**The Kinch Computer Company**  
PUBLISHERS OF TURBOTEX  
501 South Meadow Street  
Ithaca, New York 14850  
Telephone (607) 273-0222  
FAX (607) 273-0484

Helvetica, Palatino, Times, and New Century Schoolbook are trademarks of Allied Linotype Co. ITC Avant Garde, ITC Bookman, ITC Zapf Chancery, and ITC Zapf Dingbats are registered trademarks of International Typeface Corporation. PostScript is a registered trademark of Adobe Systems Incorporated. The owners of these trademarks and Adobe Systems, Inc. are not the authors, publishers, or licensors of the AP-T<sub>E</sub>X fonts. Kinch Computer Company is the sole author of the AP-T<sub>E</sub>X fonts, and has operated independently of the trademark owners and Adobe Systems, Inc. in publishing this software. Any reference in the AP-T<sub>E</sub>X font software or in this advertisement to these trademarks is solely for software compatibility or product comparison. LaserJet and DeskJet are trademarks of Hewlett-Packard Corporation. T<sub>E</sub>X is a trademark of the American Math Society. TurboT<sub>E</sub>X and AP-T<sub>E</sub>X are trademarks of Kinch Computer Company. Prices and specifications subject to change without notice. Revised October 9, 1990.

---

<b>Avant Garde</b>	Bold
<b>Avant Garde</b>	Bold Oblique
<b>Avant Garde</b>	Demibold
<b>Avant Garde</b>	Demibold Oblique
<b>Bookman</b>	Light
<b>Bookman</b>	Light Italic
<b>Bookman</b>	Demibold
<b>Bookman</b>	Demibold Italic
<b>Courier</b>	
<b>Courier</b>	Oblique
<b>Courier</b>	Bold
<b>Courier</b>	Bold Oblique
<b>Helvetica</b>	
<b>Helvetica</b>	Oblique
<b>Helvetica</b>	Bold
<b>Helvetica</b>	Bold Oblique
<b>Helvetica Narrow</b>	
<b>Helvetica Narrow</b>	Oblique
<b>Helvetica Narrow</b>	Bold
<b>Helvetica Narrow</b>	Bold Oblique
<b>Schoolbook</b>	New Century Roman
<b>Schoolbook</b>	New Century Italic
<b>Schoolbook</b>	New Century Bold
<b>Schoolbook</b>	New Century Bold Italic
<b>Palatino</b>	Roman
<b>Palatino</b>	Italic
<b>Palatino</b>	Bold
<b>Palatino</b>	Bold Italic
<b>Times</b>	Roman
<b>Times</b>	Italic
<b>Times</b>	Bold
<b>Times</b>	Bold Italic
<b>Zapf Chancery</b>	Medium Italic
<b>Symbol</b>	ΔΦΓΘΛΠΘ
<b>Zapf Dingbats</b>	%

---

# VECTOR TEX

B  
T  
w a  
g  
C & H  
E A  
M

## TEX FOR THE 90'S

**Are you still  
struggling with  
PXL's, PK's or GF's?  
Move on to scalable  
fonts:**

- Save megabytes of storage—entire VT<sub>EX</sub> fits on one floppy.
- Instantly generate any font in any size and in any variation from 5 to 100 points.
- Standard font effects include compression, slant, smallcaps, outline, shading and shadow.  
New: landscape. New: scalable graphics.
- Discover the universe of MicroPress Font Library professional typefaces: not available from any other T<sub>EX</sub> vendor.

List price \$299

Includes the VT<sub>EX</sub> typesetter (superset of T<sub>EX</sub>), 10 scalable typefaces, VVIEW (arbitrary magnification on EGA, CGA, VGA, Hercules, AT&T), VLASER (HP LaserJet), VPOST (PostScript), VDOT (Epson, Panasonic, NEC, Toshiba, Proprietary, Star, DeskJet) and manuals.

S/H add \$5. COD add \$5.

WordPerfect Interface add \$100. Site licenses available.  
Dealers' inquiries welcome. Professional typefaces available for older implementations of T<sub>EX</sub>.



**MicroPress Inc.**

68-30 Harrow Street, Forest Hills, NY 11375

Tel: (718) 575-1816 Fax: (718) 575-8038

# New **TEX** Software

American Mathematical Society

## **NEW SOFTWARE RELEASES**

The new release of ***AMS-TEX* 2.1**, ***AMS-LATEX* 1.1**, ***AMSFonts 2.1***, and the Metafont Sources for all ***AMSFonts*** are **FREE** through the AMS electronic service e-MATH or on diskettes at **NEW REDUCED PRICES!**

### ***AMS-TEX* 2.1**

A new file, amspptl.tex, has been added to provide backward compatibility with documents written under amsppt.sty version 1 but run with ***AMS-TEX* 2.0** or later.

### ***AMS-LATEX* 1.1**

***AMS-LATEX*** provides advanced mathematics typesetting capabilities to a user familiar with the **LATEX** environment. The complete Mittelbach/Schoepf font selection scheme in the /ams/amslatex/fontsel area is now available.

### ***AMSFonts 2.1***

The 2.1 version of ***AMSFonts*** has several enhancements. Euler and extra CM Fonts are now more bold and extended. ***AMSFonts*** are available in the following resolutions: 118, 180, 240, 300, and 400 dpi.

## **CUSTOMER SUPPORT**

### ***TEX Archive on e-MATH***

Technical Support Group  
American Mathematical Society  
201 Charles Street  
P.O. Box 6248  
Providence, RI 02940 USA  
(800) 321-4AMS (321-4267) ext. 4080  
(401) 455-4080  
Internet: tech-support@math.ams.com

### **e-MATH Access**

Internet: support@e-math.ams.com

### **On Diskettes**

The software files can be ordered from the AMS on Macintosh or IBM high-density, 5.25" diskettes (3.5 or low-density diskettes may be special-ordered).

Prices include shipping (first class domestic; airmail overseas).

- Prepayment is required
- Prices include shipping (by first class, domestic; airmail overseas)
- Please add 7% GST to all orders being shipped to Canada.
- For charge card orders:** TeX Library, American Mathematical Society, P.O. Box 6248, Providence, RI 02940, or call 800-321-4AMS or e-mail (Internet) cust-serv@math.ams.com
- Prepaid Orders:** American Mathematical Society, P.O. Box 1571, Annex Station, Providence, RI 02901-1571

## **ACCESS/AVAILABILITY**

### **e-MATH Electronic transfer on Internet**

AMS-developed **TEX** software files have been posted to the **TEX** archives on the Internet node e-math.ams.com (130.44.1.100) and are available for retrieval by anonymous FTP (file transfer), by those users who are on Internet. Metafont sources for all ***AMSFonts*** and guidelines for preparing electronic manuscripts in ***AMS-TEX*** and in ***AMS-LATEX*** are also on e-MATH.

All macro files and fonts in the e-MATH archive can be used with any standard implementation of **TEX**, regardless of the computer that **TEX** is running on (micros, workstations, mainframes, etc.). [A special version of ***AMSFonts 2.1*** for use with textures on the Macintosh will be added to e-MATH in the near future].

Announcements of future incremental or major upgrades will be posted to the normal **TEX** discussion lists. For more information contact support@e-math.ams.com.

## **ORDERING INFORMATION**

Upgrade diskettes for ***AMS-TEX*** and ***AMSFonts*** will be available to users of version 2.0 free through February, with a shipping charge of \$8.00.

<b><u>IBM Diskettes, 5.25", High-density</u></b>		<b>List</b>	<b>Member</b>
TEX/20	<b><i>AMS-T<small>E</small>X</i></b>	\$15	\$13
TEX/31	AMSFonts, 118 dpi	\$25	\$22
TEX/32	AMSFonts for 180 dpi printers	\$25	\$22
TEX/33	AMSFonts for 240 dpi printers	\$25	\$22
TEX/34	AMSFonts for 300 dpi printers	\$25	\$22
TEX/61	MathSci <small>T<small>E</small>X</small> —Dialog records (IBM Version)	\$15	\$13
TEX/62	MathSci <small>T<small>E</small>X</small> —CD-ROM records (IBM Version)	\$15	\$13
TEX/66	<b><i>AMS-L<small>A</small>T<small>E</small>X</i></b>	\$15	\$13
TEX/68	Metafont Sources for AMSFonts	\$15	\$13

### **Macintosh Diskettes**

TEX/58	Standard AMSFonts (magsteps 0–1)	\$25	\$22
TEX/59	Extended AMSFonts (magsteps 0–5)	\$25	\$22
TEX/60	<b><i>AMS-T<small>E</small>X</i></b>	\$15	\$13
TEX/67	<b><i>AMS-L<small>A</small>T<small>E</small>X</i></b>	\$15	\$13

## Everything You Need At One Low Price...

# Announcing the New PC<sub>T</sub>E<sub>X</sub> Systems!

You can now receive a new PC<sub>T</sub>E<sub>X</sub> System, which includes PC<sub>T</sub>E<sub>X</sub>/386 plus a full set of printer drivers, complete with everything you need to create the highest quality typeset documents possible using a PC, all at one low price. We offer a **20% Discount to TUG Members**. Here are your choices:

### The PC<sub>T</sub>E<sub>X</sub> System for Laser Printers Includes:

- PC<sub>T</sub>E<sub>X</sub>
- PC<sub>T</sub>E<sub>X</sub>/386
- PTI View
- PTI Laser/HP
- PTI Laser/PS
- PTI Jet
- CM 300dpi Fonts

Retail: \$599

**TUG Members: \$479**

### The Big PC<sub>T</sub>E<sub>X</sub> System for Laser Printers Includes:

- PC<sub>T</sub>E<sub>X</sub>
- Big PC<sub>T</sub>E<sub>X</sub>/386
- PTI View
- PTI Laser/HP
- PTI Laser/PS
- PTI Jet
- CM 300dpi Fonts

Retail: \$699

**TUG Members: \$559**

### The PC<sub>T</sub>E<sub>X</sub> System for Dot Matrix Printers Includes:

- PC<sub>T</sub>E<sub>X</sub>
- PC<sub>T</sub>E<sub>X</sub>/386
- PTI View
- PTI Dot/FX
- PTI Dot/LQ
- CM 240dpi & 180dpi Fonts

Retail: \$499

**TUG Members: \$399**

## Upgrade your Current Products and Get a Full Set of Printer Drivers, plus PC<sub>T</sub>E<sub>X</sub>/386, for only \$195

For those of you who already own PC<sub>T</sub>E<sub>X</sub>, PTI View, and at least one PTI Printer Driver, special System Upgrades\* are available to you as follows:

**PC<sub>T</sub>E<sub>X</sub> Laser System Upgrade - \$195**

**Big PC<sub>T</sub>E<sub>X</sub> Laser System Upgrade - \$245**

**PC<sub>T</sub>E<sub>X</sub> Dot Matrix System Upgrade - \$175**

## One Stop Shopping from Personal T<sub>E</sub>X, Inc.

We offer you a full range of T<sub>E</sub>X products to meet your every need... including graphics programs, fonts, spell-checkers, text editors, and T<sub>E</sub>X macros. Look for our new L<sub>A</sub>T<sub>E</sub>X book, *L<sub>A</sub>T<sub>E</sub>X for Everyone*, coming soon. For our free 1991 Product Catalog, demo diskette, or for further information, **call us today at (415) 388-8853**.

PERSONAL  
**T<sub>E</sub>X**  
INC

**12 Madrona Avenue • Mill Valley, CA 94941 • Phone: (415) 388-8853 • Fax: (415) 388-8865**

In Europe: (31) 703237241 • (49) 24167001 • (49) 80248011 • (49) 73126932 • (44) 742351489 • (39) 290091773  
(33) 169073688 • In Asia: (886) 35335179 • In Australia: (61) 34599671

\* You must provide proof of prior purchase of PC<sub>T</sub>E<sub>X</sub>, PTI View, and a PTI Printer Driver. Upgrades do not include CM Fonts. PC<sub>T</sub>E<sub>X</sub> is a registered TM of Personal T<sub>E</sub>X, Inc. T<sub>E</sub>X is an American Mathematical Society TM. Site licenses available to qualified organizations. Inquire about PTI distributorships. This ad was typeset using PC<sub>T</sub>E<sub>X</sub> and Bitstream Fonts.

# TYPESETTING: JUST \$2.50 PER PAGE!

Send us your T<sub>E</sub>X DVI files and we will typeset your material at 2000 dpi on quality photographic paper — \$2.50 per page!

Choose from these available fonts: Computer Modern, Bitstream Fontware™, and any METAFONT fonts. (For each METAFONT font used other than Computer Modern, \$15 setup is charged. This ad was composed with PCT<sub>E</sub>X® and Bitstream Dutch (Times Roman) fonts, and printed on RC paper at 2000 dpi with the Chelgraph IBX typesetter.)

And the good news is: just \$2.50 per page, \$2.25 each for 100+ pages, \$2.00 each for 500+ pages! Laser proofs \$.50 per page. (\$25 minimum on all jobs.)

Call or write today for complete information, sample prints, and our order form. **TYPE 2000, 16 Madrona Avenue, Mill Valley, CA 94941. Phone 415/388-8873.**

---

TYPE  
2000

---



In fact, it's created by the same scientists who brought you T<sub>3</sub><sup>TM</sup>, TCI Software Research Inc.

**Scientific Word**<sup>TM</sup> is the latest in PC word processing for Windows 3.0.

The file storage format is T<sub>EX</sub>. It's a full document editor, not a previewer. You compose and edit directly on the screen without being forced to think in T<sub>EX</sub>.

Your input is mathematics, and your output is T<sub>EX</sub>.

Discover the genius when you combine the power of T<sub>EX</sub> with the simplicity of **Scientific Word**<sup>TM</sup>.

To be a part of this exciting new discovery, contact TCI Software Research Inc. Call today, toll free **1-800-874-2383** for more information.



**SOFTWARE RESEARCH, INC.**

1190 FOSTER ROAD  
LAS CRUCES, NM  
88001

1-800-874-2383  
TEL: (505) 522-4600  
FAX: (505) 522-0116

T<sub>3</sub> and Scientific Word are trademarks of TCI Software Research Inc. T<sub>EX</sub> is a trademark of the American Mathematical Society. Windows is a trademark of Microsoft.



## Publishing Services



### From the Basic

The American Mathematical Society can offer you a basic TeX publishing service. You provide the DVI file and we will produce typeset pages using an Autologic APS Micro-5 phototypesetter. The low cost is basic too: only \$5 per page for the first 100 pages; \$2.50 per page for additional pages, with a \$30 minimum. Quick turnaround is important to you and us ... a manuscript up to 500 pages can be back in your hands in just one week or less.

### To the Complex

As a full service TeX publisher, you can look to the American Mathematical Society as a single source for all your publishing needs.

**Macro-Writing**

**TeX Problem Solving**

**Autologic Fonts**

**Keyboarding**

**Art and Pasteup**

**Camera Work**

**Printing**

**Binding**

For more information or to schedule a job, please contact Regina Girouard, American Mathematical Society, P.O. Box 6248, Providence, RI 02940 or call 401-455-4060 or 800-321-4AMS in the continental U.S.

### VALUABLE ADDITIONS TO YOUR TeX TOOLBOX

#### CAPTURE

Capture graphics generated by CAD, circuit design, data plotters, and other application programs that support the LaserJet. Make LaserJet images compatible with TeX. Create pk files from pcl or pcx files.      \$115.00

#### texpic

With texpic graphics package, you have the tools to integrate simple graphics—boxes, circles, ellipses, lines, arrows—into your TeX documents. Maintains output device independence.

\$79.00

#### Voyager

Macros to produce viewgraphs quickly and easily using TeX. They provide format, indentation, font, and spacing control. Macros included to produce vertical and horizontal bar charts.

\$25.00



Micro Programs Inc. 251 Jackson Ave. Syosset, NY 11791 (516) 921-1351

# For T<sub>E</sub>X Users . . .

## New Services and Prices from Computer Composition Corporation

We are pleased to announce the installation of several **new output services** now available to T<sub>E</sub>X users:

1. High Resolution Laser Imaging (1200 dpi) from Postscript diskette files created on either Mac- or PC-based systems.
2. High Resolution Laser Imaging (960 dpi) from DVI magnetic tape or diskette files using a variety of typefaces in addition to the Computer Modern typeface family.
3. High quality laser page proofs at 480 dpi.
4. **NEW PRICING** for high resolution laser imaging:
  - a. From **Postscript text files** in volumes over 400 pages . . . . . **\$2.00 per page**
  - b. From **Postscript text files** in volumes between 100 & 400 pages . . . . . **\$2.25 per page**
  - c. From **Postscript text files** in volumes below 100 pages . . . . . **\$2.40 per page**
  - d. From **DVI files** in volumes over 400 pages . . . . . **\$2.15 per page**
  - e. From **DVI files** in volumes between 100 & 400 pages . . . . . **\$2.30 per page**
  - f. From **DVI files** in volumes below 100 pages . . . . . **\$2.45 per page**

NOTE: DEDUCT \$1.00 FROM THE ABOVE PRICES FOR HIGH QUALITY LASER PAGE PROOFS.

5. **All jobs shipped within 48 hours.**

Call or write for page samples or send us your file and we will image it on the output unit of your choice.



---

**COMPUTER COMPOSITION CORPORATION**

1401 West Girard Avenue • Madison Heights, MI 48071

**(313) 545-4330    FAX (313) 544-1611**

— Since 1970 —

# A complete TeX solution that implements all the new TeX 3.0 capabilities, virtual fonts, and the Extended Font standard adopted at the TUG '90 meeting in Cork.

*ArborText put it all together. You don't have to!*

**ArborText's TeX 3.14 provides everything you need in a complete, ready-to-use package:**

Utilize the Extended TeX Font Encoding capability with pre-built virtual fonts for Computer Modern and PostScript

Use the conversion utilities we supply to make your own extended fonts from existing TeX 2.0 style fonts

Easily accent characters from your foreign language keyboard

Create multi-language documents

Choose from included hyphenation patterns for English, French, German, Dutch, Spanish, Portuguese, or add your own

Use the extended version of Plain TeX and L<sup>A</sup>T<sub>E</sub>X

We've provided access to the New Extended Fonts directly—  
*macro source included!*

TeX 3.14 and support software is available for Sun, IBM RS6000  
DEC/Risc-Ultronix, HP 9000, and IBM PCs,



ARBORTEXT INC. 535 West William Street Ann Arbor, MI 48103 FAX (313) 996-3573 Phone (313) 996-3566

DVIWindo® Previewer [y&yneward.dvi] counter[0]: 1

File Preferences Previous! Next! Unmagnify! Magnify! Help

**DVIWindo®** TM  
The Windows based DVI previewer that displays both Computer Modern (CM) and non-CM fonts.

**Now you can view documents calling for *any* outline font...**  
 Outline fonts can be scaled arbitrarily, not just to a small number of fixed sizes.  
**...and see inserted EPSF figures on screen,**  
 DVIWindo supports the ten most popular figure insertion schemes.  
**...with standard Windows interface convenience.**  
 DVIWindo provides string search, printing of page viewed, and much more.

**DVIWindo™ comes with a number of convenient utilities and now also Adobe Type Manager™**

**DVIPSONE®** TM  
The DVI converter that produces resolution-independent PostScript output using outline fonts.

**Now you can *really* exploit outline font technology in TeX...**  
 Outline font support enables use of fonts other than Computer Modern.  
**...without running out of printer memory,**  
*Partial font downloading* is used to conserve precious printer memory.  
**...while enjoying considerable cost savings.**  
 Resolution-independent PostScript files can be imaged for \$2-\$3 per page.

**DVIPSONE™ runs on IBM PC™ compatibles and can use any Adobe Type 1™ outline font.**

The PC versions of the Blue Sky Research Computer Modern Type 1 outline fonts from Y&Y now come with Adobe Type Manager. LATEX + SLATEX fonts are also available in outline form. All of these fonts can, of course, be used by any Windows application. It is by means of outline fonts that DVIPSONE and DVIWindo provide bitmap-free support for TeX!



Y&Y, 106 Indian Hill, Carlisle, MA 01741 USA  
 Telephone: (508) 371-3286 Fax: (508) 369-4445