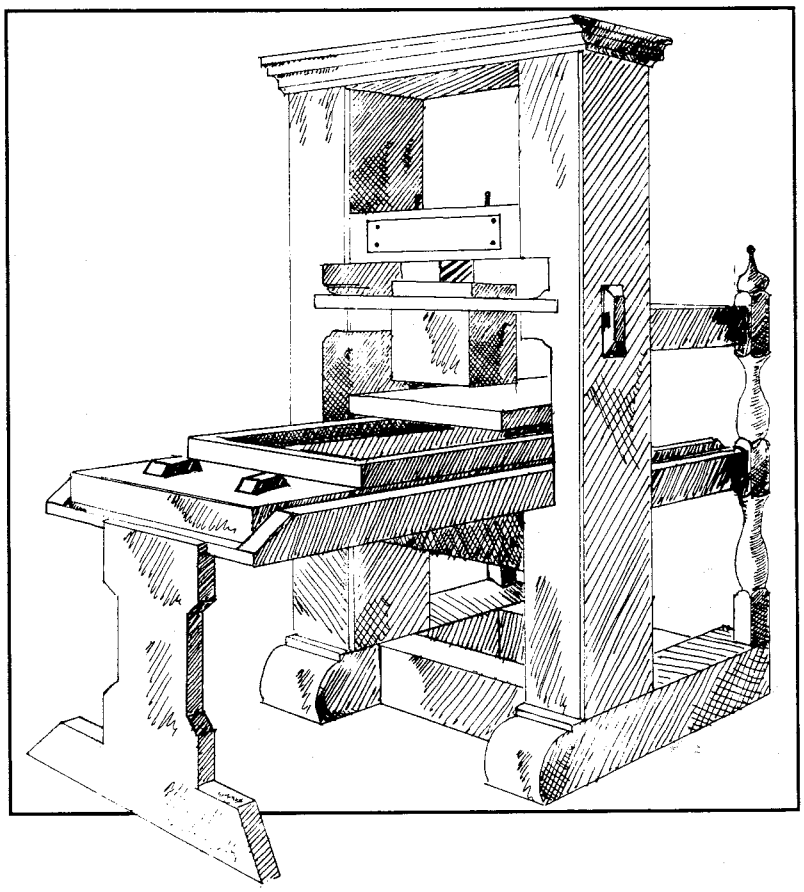
TUGBOAT

The Communications of the TEX Users Group



Volume 13, Number 3, October 1992
1992 TUG Conference Proceedings

TEX Users Group

Memberships and Subscriptions

TUGboat (ISSN 0896-3207) is published four times a year plus one supplement by the TEX Users Group, 653 North Main Street, P. O. Box 9506, Providence, RI 02940, U.S.A.

1992 dues for individual members are as follows:

- Ordinary members: $60
- Students: $50

Second-class postage paid at Providence, RI, and additional mailing offices. Postmaster: Send address changes to the TEX Users Group, P. O. Box 9506, Providence, RI 02940, U.S.A.

Membership in the TEX Users Group is for the calendar year, and includes all issues of *TUGboat* and *TEX & TUG News* for the year in which membership begins or is renewed. Individual membership is open only to named individuals, and carries with it such rights and responsibilities as voting in the annual election.

TUGboat subscriptions are available to organizations and others wishing to receive *TUGboat* in a name other than that of an individual. Subscription rates: North America $60 a year; all other countries, delivery by surface mail $60, by air mail $80.

Institutional Membership

Institutional Membership is a means of showing continuing interest in and support for both TEX and the TEX Users Group. For further information, contact the TUG office.

Addresses

General correspondence:
TEX Users Group
P. O. Box 9506
Providence, RI 02940

Payments:
TEX Users Group
P. O. Box 594
Providence, RI 02901

Parcel post, delivery services:
TEX Users Group
653 North Main Street
Providence, RI 02904

Telephone

401-751-7760

Fax

401-751-1071

Electronic Mail (Internet)

General correspondence:
`TUG@Math.AMS.org`

Submissions to *TUGboat*:
`TUGboat@Math.AMS.org`

1992 Annual Meeting Proceedings

TEX Users Group
Thirteenth Annual Meeting
Portland, Oregon, July 27–30, 1992

TUGBOAT

COMMUNICATIONS OF THE TEX USERS GROUP
TUGBOAT EDITOR BARBARA BEETON
PROCEEDINGS EDITOR MIMI BURBANK

VOLUME 13, NUMBER 3 • OCTOBER 1992
PROVIDENCE • RHODE ISLAND • U.S.A.

Production Notes

Many thanks are given to the editorial team which tackled proof-reading and copy-editing for these *Proceedings*. In addition to *Proceedings* Editor Mimi Burbank, this team consisted of Anita Hoover and Christina Thiele.

The TEX source code for each article in this issue of *TUGboat* was transmitted via network or DOS floppy diskette to the editors who then used Northlake Software's TEX running under VMS or Unix TEX to generate `dvi` files. All `\special` commands were converted to appropriate form for Radical Eye Software's `dvips`, and proof output was printed on an Apple LaserWriter NTX. Although color PostScript was supplied by at least one author, such files could not be handled by the editors (apparently) because file record lengths were too long.

`dvi` and PostScript graphics files were FTP'd to the American Mathematical Society, and final copy was produced on the Society's APS μ-5 and Agfa-Compugraphic 9600. The density of type produced on these two machines was not identical, and this was the source of some mismatches of "color" readers may see on the printed pages.

No figures were cut into pages; all were produced by incorporating 300dpi images into PostScript files.

Trademarks

Many trademarked names appear in the pages of *TUGboat*. If there is any question about whether a name is or is not a trademark, prudence dictates that it should be treated as if it is. The following list of trademarks which appear in this issue may not be complete.

APS μ5 is a trademark of Autologic, Inc.

DOS and MS/DOS are trademarks of MicroSoft Corporation

LaserJet, PCL, and DeskJet are trademarks of Hewlett-Packard, Inc.

METAFONT is a trademark of Addison-Wesley Inc.

PC TEX is a registered trademark of Personal TEX, Inc.

PostScript is a trademark of Adobe Systems, Inc.

TEX and $\mathcal{A}\mathcal{M}\mathcal{S}$-TEX are trademarks of the American Mathematical Society.

UNIX is a trademark of AT&T Bell Laboratories.

Other Conference Proceedings

Europe

Proceedings of the First European Conference on TEX for Scientific Documentation. Dario Lucarella, ed. Reading, Mass.: Addison-Wesley, 1985. [16–17 May 1985, Como, Italy.]

Proceedings of the Second European Conference on TEX for Scientific Documentation. Jacques Désarménien, ed. Berlin: Springer-Verlag, 1986. [19–21 June 1986, Strasbourg, France.]

TEX88 Conference Proceedings. Malcolm Clark, ed. Chichester, England: Ellis Horwood, 1990. [18–20 July 1988, Exeter University, Exeter, England.]

TEX90 Conference Proceedings. Mary Guenther, ed. *TUGboat* 12, no. 1. Providence, Rhode Island: TEX Users Group, 1991. [10–13 September 1990, University College, Cork, Ireland.]

North America

Conference Proceedings: TEX Users Group Eighth Annual Meeting. Dean Guenther, ed. *TEXniques* No. 5. Providence, Rhode Island: TEX Users Group, 1988. [24–26 August 1987, University of Washington, Seattle, Washington.]

Conference Proceedings: TEX Users Group Ninth Annual Meeting. Christina Thiele, ed. *TEXniques* No. 7. Providence, Rhode Island: TEX Users Group, 1988. [22–24 August 1988, McGill University, Montréal, Canada.]

Conference Proceedings: TEX Users Group Tenth Annual Meeting. Christina Thiele, ed. *TUGboat* 10, no. 4. Providence, Rhode Island: TEX Users Group, 1989. [20–23 August 1989, Stanford University, Stanford, California.]

1990 Annual Meeting Proceedings: TEX Users Group Eleventh Annual Meeting. Lincoln Durst, ed. *TUGboat* 11, no. 3. Providence, Rhode Island: TEX Users Group, 1990. [18–20 June 1990, Texas A&M University, College Station, Texas.]

1991 Annual Meeting Proceedings: TEX Users Group Twelfth Annual Meeting. Hope Hamilton, ed. *TUGboat* 12, nos. 3 & 4. Providence, Rhode Island: TEX Users Group, 1991. [15–18 July 1991, Dedham, Massachusetts.]

Introduction

Malcolm Clark

After the Annual Meeting there is the illusion that it is possible to sit back and let events unfold, secure in the warm and fuzzy glow of a successful event. Needless to say, this is indeed an illusion, and there is much still to do before my period of office ends.

The Portland Conference seemed to go well. It mostly ran to time and there was something for everyone. Yes, there were flaws, but any human activity will have flaws. A minor mistake was to forget to ask attendees to tell us what they thought of the conference. Please feel free to say what you liked and disliked about the meeting. If you didn't come, you might like to tell us why you didn't. Is it too expensive, too esoteric, too far away, too long, dull, uninteresting...?

One interesting activity pursued at the meeting was to encourage people to put themselves forward as candidates for the post of Executive Director. The search seems to be progressing well (perhaps the post will be filled by the time you read this), and we on the search committee have every confidence TUG will be in the capable hands of a new ED very soon. The candidates we talked to at Portland were excellent (as you would expect from the TEX community!).

I won't dwell on all the individual papers at the conference, but I will commend T V Raman's talk to you: 'An Audio View of TEX Documents'. He is pushing TEX in a direction which was rather unanticipated. Besides being of great direct practical importance, almost as a by product his work explores aspects of document structure which we have hardly considered before. It was for this reason that we decided to award him a special prize, in recognition of his remarkable and unique work.

The main theme of the conference was graphics. This was well supported by a large number of papers, and naturally a large number of illustrations. There was plenty else there too. If you wanted to get down to the nitty gritty of M68000 assembler code, you could listen to Barry Smith; if you wanted to program at a rather higher level, Bart Childs was there to help you with WEB; and lots of standards were lurking – PostScript, X-windows, SGML (and (LA)TEX). The Panels were lively (and I would single out Art Ogawa's contribution); the workshops I attended were very informative and productive. The vendors exhibition also made a real and useful contribution to the conference. And at last I had the chance to see an implementation of Japanese TEX, thanks to Harumi Fujiura. Excellent.

But what about the social events? We all know that the function of a conference is to bring people together, and that the talks, workshops and all the rest are merely convenient hooks. The real life is in the informal events: I can't single out one single event, they all had some special sparkle, but the Lucky 13 Dinner Party gave us a remarkable panorama of Portland (from the 41st floor of the US Bancorp Tower), and the LATEX3 bowling fundraiser had the twin advantages of an excellent evening and raising over $700 for the project. And in the background we were never far from one of Portland's gifts to west coast culture, micro-breweries. Somehow we never made it to the other advantage of Portland, hot tubs.

One of the great joys of being President is handing out prizes and honours. One of my tasks was to announce the award of a framed Bibby cartoon to a number of the Board members who retired last year, as a small token of appreciation for the work and effort that they had contributed to the organisation. These former Board members are:

Nelson Beebe	Allen Dyer	Dean Guenther	David Kratzer
Lance Carnes	David Fuchs	Hope Hamilton	Pierre MacKay
Bart Childs	Regina Girouard	Patrick Ion	Craig Platt
John Crawford			

In a more light hearted vein, Doug Henderson donated a number of prizes for the LaTeX3 fundraiser, and allowed me the pleasure of presenting them. The other set of thanks goes to all those who helped to organise the Conference locally. The staff at ETP Services aided us enormously (and we single out Mimi Jett, Dena Kaufman and Dan Olson); as did the staff at Blue Sky Research (especially Doug Henderson, Becky Kaluza and Warren Leach). Pat Rau of Northlake Software assembled a complete and informative dining guide for the city with more restaurants per capita than any other in the US. And, of course, the TUG office (Karen Butler, Cliff Alper, Teresa Pires and Kathy Sheely) provided a good deal of support as well. In addition a number of meeting participants volunteered to help wherever they could and their efforts were much appreciated.

Next year's annual conference will be at Aston University in Birmingham, UK. I hope they can find a bowling alley.

◇ Malcolm Clark
Information Resource Services
University of Westminster
115 New Cavendish Street
London W1M 8JS, UK
malcolmc@sun.pcl.ac.uk

KEYNOTE ADDRESS: Portable Graphics in TEX

Malcolm Clark
Information Resource Services
Polytechnic of Central London
115 New Cavendish Street
London W1M 8JS England
Phone: 44-71-269-3555 x3567
Internet: malcolmc@mole.pcl.ac.uk

To a very large extent, TEX was designed for the placement of characters on a page. It was implicitly assumed that the characters were probably alphabetic or mathematical. Nevertheless, Knuth notes

> If you enjoy fooling around making pictures, instead of typesetting ordinary text, TEX will be a source of endless frustration/amusement for you, because almost anything is possible...

While it is well able to draw horizontal and vertical lines, or even to plot dots more or less at random (see, for example, Knuth, 1986, p.389, and Figure 1), most people expect a little more from their graphics. There is also an architectural limitation: although TEX could easily simulate an arbitrary continuous curve by placing a very large number of small dots (or rules) on the page (or screen), TEX was only granted a finite memory. You quickly run out of memory. This is all the more distressing since there now exist versions of TEX with small and large amounts of memory (basically related to the addressing ability: 64-bit TEX on a Cray has potentially much more memory than 16-bit TEX on a pc: TEX-in-UNIX is generally somewhere in between). Sadly, this has had the effect of making TEX documents *less* portable, and seriously undermines TEX's claim to universality. TEX is universal, but the documents may be restricted to certain versions — and you won't necessarily know until you try to process them (and run out of memory, or not).

This article is reprinted from Chapter 17 of *A Plain TEX Primer*, by Malcolm Clark, Oxford University Press, 480pp, November, 1992.

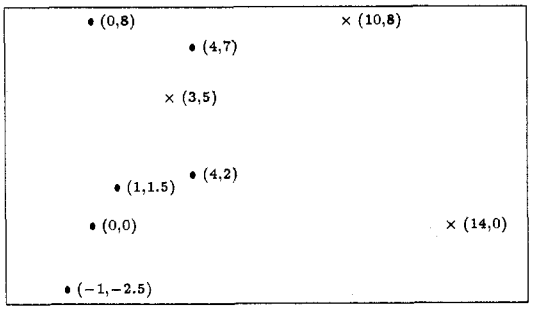


Figure 1. Simple graphics within TEX

All sorts of diagrams have been created using TEX. References to some of these are given in the bibliography.

There are three major ways in which graphics may be made part of TEX documents. For simplicity and brevity, 'graphics' is restricted principally to line graphics, but most of what is covered can be generalized. As with most things, the more limited the capabilities, the closer they may be to universality. High degrees of sophistication usually mean greater restrictions are present. Attention is directed here to techniques which have some claim to generality: the 'running on my Sun workstation using proprietary software' solution is ignored as far as possible. The vain hope is that someone working on their Macintosh will be able to exchange TEX documents with someone working on an IBM pc, an Amiga, an Atari, a NeXT, a Vax under VMS, and so on up the scale until we reach the supercomputer league. We do not wish to present solutions which only work on specific boxes. UNIX may be the *de facto* operating system, just as PostScript is the *de facto* page description language. But there are more non-UNIX boxes out in the world than there

are UNIX boxes. Similarly, there are more non-PostScript output devices than there are PostScript outputs. If everybody were to standardize on the same computing box many problems of interchange would go away, but this is unlikely to happen.

Special Fonts

This first approach is limited, but very general — it will work with TEX and any of its drivers. It is possible to use special fonts to build pictures. Again there are three main ways to do this: the first is through simple font elements (that is, straight line segments, or curves) which can be assembled to give (fairly simple) pictures. The second is through METAFONT. Here, we use METAFONT to create a single character which is our graph (or whatever). This seems intimidating, but need not be. And lastly, we can create special fonts without METAFONT.

Simple font elements. So, start with the simple font elements. Knuth gives an example in *The TEXbook*, pages 389 – 391, but the font he uses is not generally available. Alternatively, LATEX already does this, in its `picture` environment (cf. Lamport, 1986, pp.101 – 111). Unfortunately Lamport did not develop this to the same extent as the rest of LATEX, and it has a distinctly 'squared graph paper' feel. But it is certainly possible to create quite attractive graphs. Any vertical and horizontal elements are just standard TEX rules, while rounded corners and circles can be made from the LATEX circle fonts — (Figure 2). A small range of diagonals is possible through other special line fonts — ////////.

The LATEX `picture` environment is amazingly modular. In other words you can rip it out of LATEX and run it in `plain` TEX, using the same basic commands which are documented in the LATEX book. Although creating pictures this way is time consuming, it can give quite pleasing quality (at least on the laser printer). Quite acceptable bar charts may be created, as Nagy (1989) shows (Figure 3). It is possible to tackle chemistry through the use of these fonts, as Figure 4 demonstrates. In this case some of the tedium is removed by creating the ring structure only once, storing it in a box, and then copying that box when it is needed. Besides making the procedure less long winded, it cuts down on the effort needed by TEX itself, since copying a box requires no new manipulations.

The creation of diagrams like this can be amazingly tedious, but the approach still achieves a generality and portability which cannot be ignored.

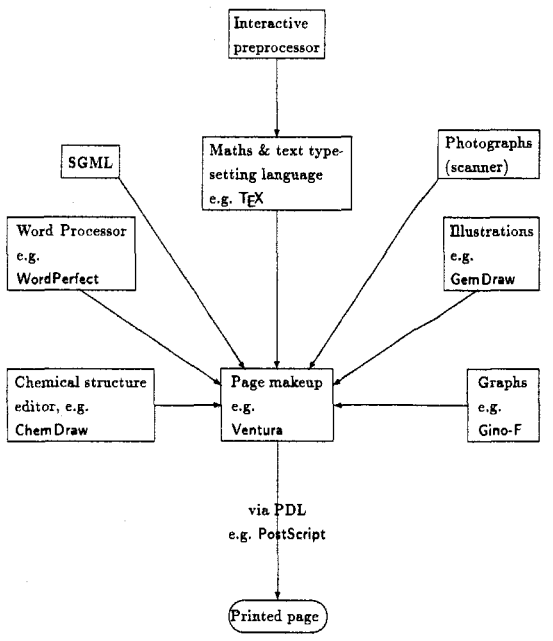


Figure 2. Using the LATEX fonts, from Norris and Oakley (1990)

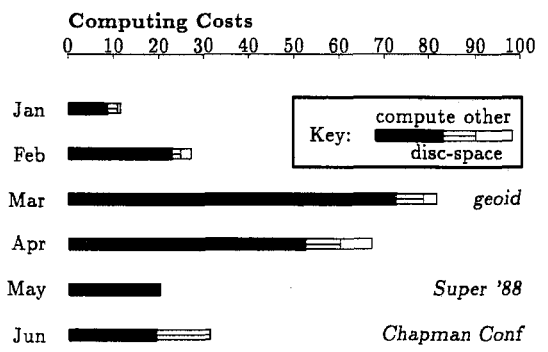


Figure 3. Bar chart, from Nagy (1989)

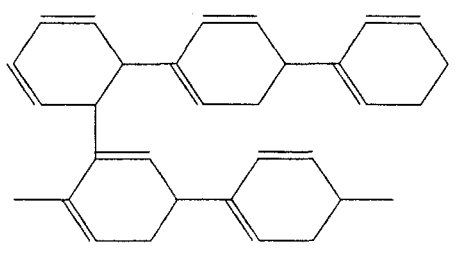


Figure 4. Simple chemistry with LATEX fonts, from de Bruin *et al.* (1988)

Because of this generality, there are some pre-

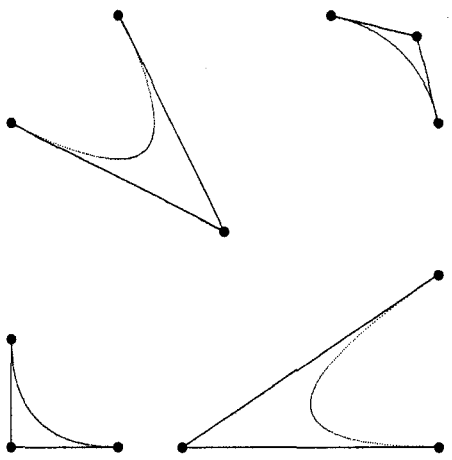


Figure 5. Bézier curves and control points, from Beebe (1989)

processor programs which will allow you to create something interactively which is then transformed into LATEX commands. If you have access to a UNIX system, **gnutex** can assist. On an MS-DOS system, part of the emTEX package does just this (although it adds a few extra features of its own). The key drawbacks of the LATEX special font approach are centered around the limited fonts which are available, both in the slope of lines and their thicknesses, and the limited range of curves. These very limited resources can be encouraged to generate quite an amazing range of possibilities. But an enormous amount of time and effort is also required. Having said this, traditionally a tremendous amount of effort had to be expended to create diagrams like these anyway. In this way we have a single document, and the opportunity to revise.

An advantage of course is that everything is in (LA)TEX, so that we can ensure that the relative weights of lines, the font sizes, the symbols, blend in well with the rest of the document. This is a feature which we should not ignore.

A further advantage is the ability to preview the diagram on the screen. Since the METAFONT descriptions of the fonts are available, the screen fonts may also be generated.

The use of the rules might indicate that you could build the most complex curves out of small rectangular boxes: make them small enough and it will not be possible to see the join. In fact, an extension to LATEX `picture` environment is the `bezier` style, which allows a bézier curve to be plotted (see Figure 5). Make too many of them and TEX runs out of memory.

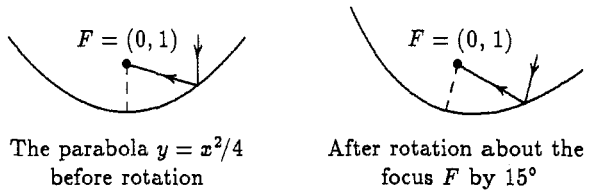


Figure 6. PICTEX graphics, from Wichura (1987)

Resolution becomes an issue if we try to create continuous curves from small elements. If TEX memory fills up quickly at 300 dpi, it will fill up even more quickly at 1270 dpi. It is difficult to claim device independence when we must take resolution into account. We can of course ignore the resolution problem, but on those times when we want to produce high-quality graphs, we may be disappointed by the faithful rendition of those 300 dpi blobs, and the angular 'staircasing' which is all too obvious at the higher resolution.

The creation of bézier curves is a remarkable achievement, given TEX's limited arithmetic capability. Adding two numbers together is awkward enough, and when we realize that TEX will only use integers in a rather limited range, the results are all the more surprising.

Since the `picture` environment is rather crude, one or two people have put higher-level commands around them. The two best known are PICTEX and `epic`. PICTEX (Wichura, 1987) can be run with both TEX and LATEX (Figure 6).

The commands for PICTEX are distributed freely, but the 85 page manual is essential in order to use it sensibly. This article has already loaded quite a few picture-drawing commands and many of the allocation registers are becoming filled up. While it is no real problem to stick to (say) the `picture` environment, once we start mixing in extra commands the limitation to 256 counters, boxes, dimensions, and token strings starts to hurt.

The syntax of the commands required by PICTEX seems quite reasonable, if quirky at times. It is no worse than many commercial plotting packages like SAS or SPSS. But even if we have enough room for allocation of the registers, running with PICTEX and LATEX, on a 32-bit TEX, it is still possible (but not easy) to exhaust the available memory. And given the amount of arithmetic going on in the background, these diagrams tend to be slow.

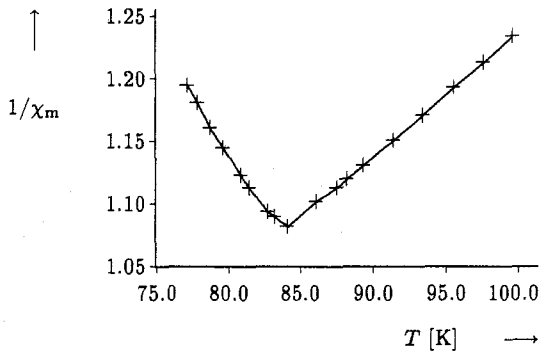


Figure 7. Reciprocal magnetic susceptibility, from Ramek (1990)

Olivier (1989) describes an amalgam between S, the UNIX statistical package, and PICTEX. Clearly this is restricted to UNIX in the first instance, although the PICTEX would be portable.

Although `epic` (Podar, 1986) was targeted for LATEX it can also be used in TEX. It lacks the generality of PICTEX, but is a useful extension. Podar added some higher-level commands in order to provide a 'friendlier and more powerful users interface'. In particular he managed to reduce the amount of manual calculation required. For example, he introduced a `\drawline` command which allows specified points to be connected. In order to avoid the problem of slope segments outside LATEX's ability, he uses the closest slope available. This can lead to rather jagged lines. If the lines are dashed, this problem appears less acute.

There are several collections of commands which draw all sorts of rather nice graphs. My favourites are those of Michael Ramek (Ramek, 1990). Figure 7 is taken from his paper and helps illustrate the scope that is possible. Besides the 'normal' graph requirements, he provided some other commands to draw chemical structures as shown in Figure 8.

Other fonts. So far we have been discussing the use of special fonts. Of course, we can also generate our own. There are two different directions here. On the one hand we can use some suite of other fonts; on the other we could generate METAFONT descriptions somehow and use those descriptions. In both cases there is appreciable generality. In the final analysis, METAFONT is as portable as TEX, and once the descriptions are made available we are

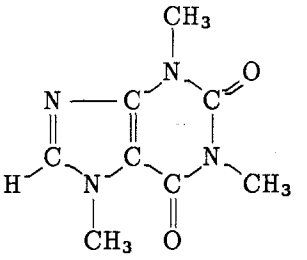


Figure 8. Caffeine

as free to use those as we would be to use (LA)TEX commands.

Knuth (1987) introduced some halftone fonts which allow greyscale 'pictures' to be typeset in a completely device independent way. Adrian Clark (1987) also made some contribution to this, and Hoenig (1989) shows some interesting examples. Since the descriptions are available, anyone may 'borrow' them quite easily. Adrian used a FORTRAN program as a pre-processor. This is fair, since for all sorts of reasons we would normally expect the data to be provided in a digital form from some other source. There are problems of TEX memory here again. Even with a 'big' version, TEX may only handle one 512 × 512 picture (or four 256 × 256 pictures). Knuth's paper discusses some manipulation techniques which would allow greater clarity from lower-resolution pictures. This is a fairly general and well-understood aspect of image processing which need not concern us here. The point is that it is quite possible and represents no new addition of hardware or software.

An alternative use of METAFONT is to view it as a means of describing an arbitrary picture, not a typeface. All the tools are there to do it, and in fact it is really a lot simpler than creating fonts. Of course, you do not really do it in METAFONT; you do it in something else, which is then translated to METAFONT. The something else at the moment is one of several programs by Rick Simpson (Simpson, 1990), which works on the IBM RT (running AIX, a UNIX lookalike), or Metaplot (Pat Wilcox, 1989). This latter was written in C, and is available in a number of forms. There is at least a pc version, an Amiga version, and lots of UNIX versions.

In both cases, what comes out at the far end is a single (very large) character (or even a set of characters which are 'tiled' together), which you plot wherever you want. The disadvantage is that scaling the picture is tedious (just like scaling a 'normal' character), and editing it requires a re-run

of METAFONT. But it is device independent. The only proviso is that the device driver be able to handle these very large characters. This is not a trivial expectation, since many drivers were written expecting that they would be dealing with letters, and that there was some reasonable maximum size to a letter.

Wilcox did not really expect the user to write in her 'Metaplot'. The notion was that a variety of other, arbitrary, plotting languages could be mapped onto the Metaplot commands, which were then shipped to METAFONT.

CGM, or Computer Graphics Metafile, is worth considering too. It has a couple of features which we ought to bear in mind. It is an international standard. Nominally, every graphics package ought to have the facility to generate CGM, and also to read it in. The metafile should also be able to be transmitted over electronic networks with the minimum of fuss. The other feature is that CALS (Computer-aided Acquisition and Logistics System) has adopted CGM as one of its components: while we may worry about the militaristic background of CALS, it has done much to revitalize and make acceptable SGML, and we can expect it to help in the adoption of CGM. One other component of CALS is that it has adopted another 'graphics' standard, IGES. IGES is usually described as a *de facto* standard; it was developed principally for use with CAD-CAM software. Nevertheless, it does offer another routeway. In essence there is no real reason why Metaplot could not read an IGES file and transform it to METAFONT form. Since we are in the real world of 'standards', Heinz (1990) notes that GKS (Graphics Kernel Standard) may also be transformed into TEX.

Another route to create a character. If we look a little more closely at what a driver actually requires to set a character, we note that there are two items: the pixel file, and the TEX font metric file. Conventionally, the route to produce these is METAFONT, but there is no particular reason why we should have to adopt this route. Provided the `tfm` and `pk` contain appropriate information, the driver should be able to typeset. The underlying idea here is that we can have another program take (say) a grey-scale picture and process it to produce both the required files. The `tfm` file should be simple enough to produce, even by hand, since we might make this 'font' have only one character at a time. The property list would be fairly simple. A traditional pixel (or `pxl`) file only contains binary information, so we are back in the realms of image processing or half toning if we wish to do something rather fancy. Most drivers now accept 'packed pixel' rather than 'pixel' information. This is simply a far more compact form of the same information.

Simpson (1990) also describes an application of this approach. The example he chooses takes a raster image and turns it into a font. The program imtopk converts an IMPART image processing file into a `pk/tfm` pair. impart handles the image scaling, allowing for device pixel density, does any filtering necessary, and converts an n-level grey scale to two levels. TEX positions the image on the page, typesets any annotation, and handles any other typesetting. At Texas A&M University, a similar approach is used where output from a number of graphics programs, but especially the graphics software package 'Dissipla', is processed to produce the `pk/tfm` pair. This has some appeal since Dissipla runs on a very wide variety of machines, and may even be called from programming languages. A drawback of this approach is that it is difficult to annotate the diagrams with fonts similar to the ones used in the TEX document.

Special

Now to the less general: any sort of material may be incorporated in a `\special`. Whatever appears there is passed directly to the `dvi` file, where it will be handled by the `dvi` driver. For example, we could have PostScript commands in there (or even a reference to a file containing a PostScript-created graphic). The problem is that you also need a driver which knows what to do with the information, and a device (printer/screen) which can display the information. While PostScript is described as a *de facto* standard, not everyone has access to a PostScript device, and in fact more Hewlett Packard (and compatible) machines are out there in the real world than anything else.

This actually opens up another route. While we could easily include a complete graphic produced by another approach (one of the vast array of graphics packages which will produce PostScript), and probably scale or otherwise modify it, we can also pass simpler information to the `dvi` file for processing by the driver. Maus and Baker (1986) extended the LATEX `picture` environment by adding a whole host of commands, which, when examined closely, are little 'specials' which do things like draw a line of arbitrary slope through PostScript commands. Now TEX does not process anything; therefore TEX's memory does not fill up. When printed (on a PostScript device), the line is there.

Unfortunately, only a few screens are PostScript devices, and so we don't usually expect to see these elements previewed.

One other disadvantage of using specials is that the form of specials is by no means standardized. Although there is a working party (TUG, 1992) attempting to standardize and issue recommendations, they are facing the usual problems of standardization committees. One of the recommendations is that a level 0 driver should be able to place at least 1000 rules and 20,000 characters on a single page, unless the output device is constrained in some way. On-board device memory may be limited and limit these ideal minima.

Recall that well over half of the drivers written for use with TEX reside in the public domain. No commercial forces come into play with them, nor can the TEX Users Group impose rules (it is there to serve its members, not police them: in general this sort of anarchy works, since there is enough goodwill around). What we are coming to is the fact that specials have to be written with a specific driver in mind. To give an example: imagine we want to ship out a couple of PostScript commands, represented by `<command>`. Using *Textures* on the Macintosh, which has its own built-in driver, you could say

```
\special{postscript <command>}
```

Using ArborText's (1987) PostScript driver, DVILASER/PS, the command is

```
\special{ps:: <command>}
```

Using the public domain DVI2PS, the structure is

```
\special{pstext=<command>}
```

or using another public domain driver DVIPS (Tom Rokicki), the equivalent is

```
\specialps:  <command> or
\specialps::  <command>
```

while Nelson Beebe's driver (Beebe, 1987) appears to have no way of including a single command (you could obviously use the facility to read in a file, which itself contained only one command); similarly, Personal TEX's PostScript driver (Personal TEX, 1987) appears to lack the 'in-line command' feature.

Trevor Darrell (1987) wrote a useful set of commands, `psfig`, which greatly ease the problems of incorporating PostScript into a document. The PostScript is really 'encapsulated', since the 'bounding box' information is required. 'Encapsulated' also implies that the PostScript should not change the state of commands — in other words, that any changes should be local (in TEX terminology). The portion of `psfig` which deals with the `\specials` is well separated, and it is possible to modify that part of the command suite for particular drivers.

You could reasonably ask why we do not include CGM files in `\specials`. In fact, this has been done (Andrews, 1989). Provided the driver can handle the commands and change them into the correct form for the output device, any sort of file can be processed. As noted earlier, the `dvi` is itself a sort of metafile. Andrews' extensions work for UNIX and VMS environments.

PostScript is not yet ubiquitous. Fortunately, there is also an approach which allows us to use a Hewlett Packard LaserJet — CAPTURE (Pickrell, 1990). Any program which produces output for a LaserJet can have that output processed with CAPTURE to produce a file which may be input to TEX, through some suitable commands (which will, somewhere, employ `\specials`). Again, this sounds longwinded, but there are a great many programs which will do this. Even more remarkable, there are programs which can take PostScript and turn it into LaserJet form (Freedom of the Press, GoScript, Ghostscript, etc.). This means that we are now relatively independent of PostScript.

In Betweens

A few years ago the notion of 'little languages' became current. This is a scheme which is found most generally in UNIX. Instead of adding features to troff, 'little languages' were created: preprocessors which massaged some reasonable form of input into troff. These include chem (for chemistry), tbl (for tables), eqn (for equations), grap (for general graphs), and pic (for pictures). The one we are interested in is pic and perhaps grap: pic has a language which allows creation of line diagrams with embedded text. Sounds simple. Of course, with the way that UNIX works, it is 'easy' to write a command line which hides all the 'little language' bits and pieces from the end user.

How is this relevant? Recall that TEX passes `\special` information straight to the `dvi` file. That information could easily be special commands which the driver could interpret. If we pass PostScript commands, then the driver can handle PostScript (maybe). What if we pass higher-level commands which the driver then processes to produce a new `dvi` file? In other words, a `dvi` to `dvi` processor. The new `dvi` file would, among other things, be able to be previewed, or be sent to any suitable printer (provided you had the correct `dvi`-to-printer

driver). So what we end up with is a *device independent* method.

There are a couple of attempts to do this. There is a program around called `dvidvi` (Rokicki, 1989) which processes a `dvi` file, but only so that you can rearrange the pages — say to shrink them to thumbnails and arrange them all on a single sheet (actually very useful for book make-up). Mike Spivak (1989) has provided `dvipaste` which allows you to 'paste' a `dvi` file into another `dvi` file, so that you can put a table (which gobbles up space in TEX) where you want (equally it could paste in a large picture — and that is why it has been mentioned here). And lastly, the one that really does pictures, Rolf Olejniczak's `texpic` (1989). This is a TEX implementation of pic which does all the things that pic does and more, and works in just the way outlined.

What is the snag? The driver has to be implemented on all sorts of different machines. We are gnawing away at the portability. Including PostScript or Hewlett Packard's laser printer language seems also eminently non-portable. At least this localizes the problem and in the longer term gives a far more general solution. Olejniczak's program is available only for MS-DOS, and is currently proprietary, although it is not especially expensive. It is the restricted platform which is the real problem.

Closing Comments

Beebe (1989), Rahtz (1989), and Heinz (1990) have all contributed to the discussion of incorporating graphics into TEX documents. The adoption of the METAFONT and `pk/tfm` solution goes some way to ensuring the transportability of documents. None of the other approaches yet comes close enough to being capable of being transmitted over fairly arbitrary networks. Another advantage of this approach should be the capability of viewing the diagrams on the screen, as well as on paper. The tools which enable these transformations ought to be part of the standard TEX distributions. Within a closed environment, any solution which works is to be applauded. But one of the major features of TEX is its 'open'-ness, and the portability of documents created with TEX.

It will have become apparent that we are always in the hands of the drivers available. This is perhaps the weakest link in the whole chain. Whether you regard the drivers as part of TEX or not depends on your viewpoint.

It is perhaps wise to remind ourselves that even in the days of Johann Gensfleisch zum Gutenberg the integration of text and illustration (through woodblocks) took some time, and could only be achieved after agreement with the professional woodblock cutters.

Bibliography

Andrews, Phil. "Integration of TEX and graphics at the Pittsburgh Supercomputing Center." *TUGboat* 10(2), pages 177–178, 1989.

ArborText. *Dvilaser/PS User Manual*, 1989.

Beebe, Nelson H.F.. "A TEX DVI Driver Family." *TEXniques* 5, pages 71–113, 1988.

Beebe, Nelson H.F. "TEX and Graphics: The State of the Problem." *Cahiers GUTenberg* 2, pages 13–53, 1989.

Bruin, Rob de, Cornelis G. van der Laan, Jan R. Luyten, and Herman F. Vogt. "Publiceren met LATEX." *CWI Syllabus* 19, 1989.

Clark, Adrian F. "Halftone Output from TEX." *TUGboat* 8(3), pages 270–274, 1987.

Clark, Adrian F. "Practical Halftoning with TEX." *TUGboat* 12(1) pages 157–165, 1991.

Darrell, Trevor. "Incorporating PostScript and Macintosh Figures in TEX." Electronic document available with `psfig` commands, 1987.

Ehrbar, Hans. "Statistical Graphics with TEX." *TUGboat* 7(3), pages 171–175, 1986.

Gehani, N. "Tutorial: UNIX Document Formatting and Typesetting." *IEEE Software*, pages 15–24, September 1986.

Gruber, H., E. Krautz, H.P. Fritzer, K. Gatterer, G. Sperka, W. Sitte, and A. Popitsch. "Electrical Resistivity, Magnetic Susceptibility, and Infrared Spectra of Superconducting $RBa_2Cu_3O_7$ with R = Y, Sc, Tm, Ho, Eu, Nd, Gd." Pages 83–88 in *High-T_c Superconductors*, H.W. Weber, ed. New York: Plenum Press, 1989.

Heinz, Alois. "Including Pictures in TEX." Pages 141–151 in *TEX Applications, Uses, Methods*, Malcolm Clark (ed.). Chichester, England: Ellis Horwood Publishers, 1990.

Hoenig, Alan. "Fractal Images with TEX." *TUGboat* 10(4), pages 491–498, 1989.

Kernighan, Brian W. "*pic* — A Language for Typesetting Graphics." *Software — Practice and Experience* 12(1), pages 1–21, 1982.

Kernighan, Brian W. "The UNIX Document Preparation Tools — A Retrospective." Pages 12–25 in *PROTEXT I*. J.J.H. Miller, ed. Dublin: Boole Press, 1984.

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

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

Knuth, Donald E. "Fonts for Digital Halftones." *TUGboat* 8(2), pages 135–160, 1987.

Lamport, Leslie. *LATEX: A Document Preparation System*. Reading, Mass.: Addison-Wesley, 1985.

Maus, Doug, and Bruce Baker. "Dvilaser/PS Extensions to LATEX." *TUGboat* 7(1), pages 41–47, 1987.

Murray, Peter, and Linda Murray. *The Art of the Renaissance*. London: Thames & Hudson, 1963.

Nicole, Olivier, "A Graphic Driver to Interface Statistical Software S with PICTEX." *TUGboat* 12(1), pages 70–73, 1990.

Norris, A.C., and A.L. Oakley. "Electronic Publishing and Chemical Text Processing." Pages 207–225 in *TEX Applications, Uses, Methods*, Malcolm Clark (ed.). Chichester, England: Ellis Horwood Publishers, 1990.

Olejniczak-Burkert, Rolf. *texpic User Manual 1.0*. Interplan TB Software GmbH., 1990.

Olejniczak-Burkert, Rolf. "*texpic*— Design and Implementation of a Picture Graphics Language in TEX à la pic." *TUGboat* 10(4), pages 627–637, 1989.

Personal TEX Inc. *PTI Laser/PS Manual*, 1987.

Pickrell, Lee S. "Combining Graphics with TEX on IBM PC-Compatible Systems and LaserJet Printers." TEX Users Group 11(1), 26–31, 1990.

Podar, Sunil. "Enhancements to the Picture Environment of LATEX." Technical Report 86-17, Department of Computer Science, SUNY, 1986.

Rahtz, Sebastian. "A Survey of TEX and Graphics." CSTR 89-7, Department of Electronics & Computer Science, University of Southampton, 1989.

Ramek, Michael. "Chemical Structure Formulas and X/Y Diagrams with TEX." Pages 227–258 in *TEX Applications, Uses, Methods*, Malcolm Clark (ed.). Chichester, England: Ellis Horwood Publishers, 1990.

Rokicki, Tom. "`dvidvi`: read.me." Electronic documentation), Radical Eye Software, 1989.

Rokicki, Tom. DVIPS: A TEX Driver.

Salomon, David. "DDA Methods in TEX." *TUGboat* 10(2), pages 207–216, 1989.

Schöpf, Rainer. "Drawing Histogram Bars inside the LATEX Picture-Environment." *TUGboat* 10(1), pages 105–107, 1989.

Simpson, Richard. "Nontraditional uses of METAFONT." Pages 259–271 in *TEX Applications, Uses, Methods*, Malcolm Clark (ed.). Chichester, England: Ellis Horwood Publishers, 1990.

Ballantyne, Michael, Michael D. Spivak, and Yoke Lee. "HI-TEX Cutting & Pasting." *TUGboat* 10(2), pages 164–165, 1989.

TUG `DVI` Driver Standards Committee. "The `DVI` Driver Standard, Level 0." *TUGboat* 13(1), pages 54–57, 1992.

Van Haagen, A.J. "Box Plots and Scatter Plots with TEX Macros." *TUGboat* 9(2), pages 189–192, 1988.

Wichura, Michael. *The PICTEX Manual*. (*TEXniques* 6 series. Providence, Rhode Island: TEX Users Group, 1987.

Wichura, Michael. "PICTEX: Macros for Drawing PICtures." *TUGboat* 9(2), pages 193–197, 1988.

Wilcox, Patricia. "Metaplot: Machine Independent Line Graphics for TEX." *TUGboat* 10(2), pages 179–187, 1989.

Wujastyk, Dominik. "Chemical Ring Macros in LATEX." *TEXline* 4, page 11, 1987.

Wujastyk, Dominik. "Chemical Symbols from LATEX." *TEXline* 5, page 10, 1987.

Literate Programming, A Practioner's View

Bart Childs
Texas A&M University
Department of Computer Science
College Station, TX 77843-3112
Phone (409) 845-5470; FAX (409) 847-8578
Internet: bart@cs.tamu.edu

Abstract

I have been using the WEB style of Literate Programming since my first efforts to port TEX to the Data General AOS system. When I looked back at those efforts, the work in porting drivers that were not written in WEB and the writing of drivers in WEB (based upon DVITYPE, of course), the value of this method of programming became evident.

I have concentrated my research (and some teaching) efforts upon this style of programming. I will relate my insights and opinions of the following: some quantitative and qualitative measures of the value of WEB programming; a description of some tools that are part of an environment for writing and maintaining literate programs; literate programming environments that are alternatives to the WEB style; an annotated list of some literate programming systems; and I will conclude with my perception of the future of literate programming.

Introduction

Donald Knuth created the WEB system of literate programming when he wrote the TEX typesetting system a second time (see "The WEB system of structured documentation", 1983; "Literate programming", 1984; *TEX: The Program*, 1986; and METAFONT: *The Program*). The WEB system can be described as the merging of documentation, code, and presenting the listings in a typeset format with aids of table of contents, cross-referencing, and indices.

I have used Knuth's original WEB system and several descendants for a number of years. It is my opinion that the training necessary to learn how to program in a literate style is relatively small. I believe the benefits of literate programming make it worthwhile. The benefits are better and more maintainable code (it can be argued that this is not proven.)

In this paper I will also report on available literate programming systems, some of my successes and failures as a literate programmer, creation of some tools to aid the literate programmer, the community of literate programmers that I have been able to identify, alternatives to the WEB system, and suggested directions for use and research in literate programming.

A Definition

I use the following list of requirements to imply a definition of a literate program and the minimum set of tools which are needed to prepare, use, and study the resulting code.

- The high-level language code and the system documentation of the program come from the same set of source files.
- The documentation and high-level language code are complementary and should address the same elements of the algorithms being written.
- The literate program should have logical subdivisions. Knuth called these *modules* or *sections*.
- The system should be presented in an order based upon logical considerations rather than syntactic constraints.
- The documentation should include an examination of alternative solutions and should suggest future maintenance problems and extensions.
- The documentation should include a description of the problem and its solution. This should include all aids such as mathematics and graphics that enhance communication of the problem statement and the understanding of its challenge.

- Cross references, indices, and different fonts for text, high-level language keywords, variable names, and literals should be reasonably automatic and obvious in the source and the documentation.

These requirements have been adapted from (Knuth, 1992), and (VanWyk, 1989 and 1990). My adaptations of the list were affected by my experience as a WEB user — first in a maintenance mode, then as an author, and finally using WEB in undergraduate and graduate education environments. The last has involved the creation of some tools to enhance the use of literate programming in all environments.

Knuth posed this thought to introduce literate programming: "Instead of imagining that our main task is to instruct a *computer* what to do, let us concentrate rather on explaining to *human beings* what we want a computer to do" (Knuth, 1984). His thesis was that it should be just as important to communicate with the other persons who read the program as it is to communicate with the computer which executes it (Knuth, 1992). David Ness said "it is the **most** important task."

Knuth's WEB System

When Don Knuth wrote TEX the second time, he gave thought to making it portable to many different systems. WEB was created as a superset of TEX and Pascal.

WEB's design encourages writing programs in small chunks which Knuth called modules (he also used the term sections). Modules have three parts: *documentation, definitions,* and *code.* At least one of these three parts must be *non-null.*

The *documentation* portion is often a verbal description of the algorithm. It may be any textual information that aids the understanding of the problem. It is not uncommon for a WEB to have a number of 'documentation only' modules. These usually describe the problem independent of the chosen language for implementation. For example, a WEB for a subprogram that solves the linear equation, $Ax = b$, could have discussion of singularity, condition numbers, partial pivoting, the banded nature of the expected coefficient matrices, etc. It should be an unusual but not exceptional case when a module contains no documentation.

The definition part of a module was often used by Knuth to offset shortcomings of Pascal when used in *systems programming.* (Wirth created Pascal to be a language for pedagogy, not systems programming.) This is rarely the language of choice by today's systems programmers but sometimes it is a convenient way to represent certain ideas. (Knuth used Pascal because "it was everybody's second best language" at the time (Knuth, *TUGboat*, 7(2), 1986). It was before C was widely available.)

Some of My Successes and Failures

My introduction to the WEB system of literate programming was rather abrupt — I was porting TEX82. I had not written a Pascal program or a TEX document at the time. I had written a number of systems programs in PL/1.

Version 0.6 of the TEX system was made available to me in the fall semester of 1982. Distribution tapes contained about 300 files. Most of these had to do with the fonts in a binary form. Binary files were written in a format of twenty bytes per record. Each byte was converted to four ASCII characters; for example, '␣255' for the byte with all bits being one. Over a two or three week period I ported the TANGLE processor and began work on the WEAVE processor. I also had the 'report version' of volume B of the *C & T* series (Knuth, 1986).

The end of the semester caused me to stop the work and I did not resume until the spring with version 0.9 of TEX. I spent some time reviewing the changes in the *necessary* parts of the TEX system: TANGLE, WEAVE, and TEX.

The WEB sources were usually complemented with the change files for TOPS, VMS, and UNIX systems. These were valuable, but I recall the feeling that the "system dependencies" entries in the index were even more valuable. It helped to see what was changed for other systems, but often those changes were in terms of "system calls" whose documentation I did not have.

The *necessary* codes are approximately 2,700, 7,000, and 25,000 lines of Pascal each of the INITeX and TEX processors. I was able to port approximately 60,000 lines of Pascal code to a system on which only 2,700 lines had previously been ported. I did this in **three days** (probably 15 hours of work). I regret that I did not keep a diary or log errors like Don Knuth did, (Knuth, *TUGboat*, 7(2), 1986; and Knuth, chapters 10 and 11, 1992).

The experience of porting the system convinced me that the literate programming style had significant merit. Most of the programs I subsequently wrote for the TEX system were WEBs. The existence of dvitype as a model for drivers helped this decision. I created a family of drivers for the AOS system for QMS, HP, and Canon printers. These used a common source and adaptations for the different printers were accomplished by change files.

This further experience led to the logical conclusion that an "environment" could help significantly. We created several environments to automate the steps of creation of code and documentation for the AOS system. The emergence of UNIX and availability of workstations has relegated these early works to simply being a pleasant memory that in some sense could be called a failure.

Marcus Brown built part of an "Interactive Environment for Literate Programming" as part of his dissertation under my direction (Brown, 1988 and 1990). This was essentially the output side of an editor which also allowed the navigation of the source with views of the typeset output, a graphical tree structure of the module interconnections, and other functionalities. It did not include a real editor. The environment was tested by giving senior computer science students the tasks of identifying the changes to make a "big TEX" and changing **tangle** to make code that is more readable. (Knuth's original WEB created code that was to be "unfit for human consumption".) These students had been using WEB in processing system performance logs.

The positive results of this work were that the subjects performed well in identifying the necessary changes using the typeset listings of the codes and in the on-line form using the environment.

The environment was dependent upon SUN graphics and did not lend itself to incorporation with public domain editors. A later environment was based upon GNU Emacs.

Other WEB Systems

There have been a number of WEB and WEB-like systems developed. They can be divided into several categories.

- WEB systems that have the same set of tools that are adapted to a different high-level language. The high-level languages supported include: C (Guntermann and Schrod, 1986 and Levy, 1987), FORTRAN (Krommes, 1989), Modula-2 (Sewell, 1987), LISP dialects, and Reduce. In the references I also indicate the sites where the 'definitive' sources are available.
  There are some differences in functionality in many of these such as support of multiple change files (Guntermann) and several high level languages (Krommes).
- WEB systems that have been adapted to a different high-level language by pre-processors and/or post-processors. These include support for FORTRAN, and the first WEB for Reduce.
- WEB systems that have a different basis for their creation but generally follow the same WEB concepts. Ramsey's Spider enables reasonably easy creation of WEBs for several different languages (Ramsey, 1989). His example include WEBs for Ada, C, awk, and SPL. Gragert and Roelofs created the second WEB for Reduce with Spider (Gragert, 1991). They are now creating a WEB for Maple (Gragert, 1992).
- WEB systems using a different language or formatting system. Three will be mentioned. Thimbelby did his CWEB with the UNIX standards of C and *troff* (Thimbelby, 1986). The limitations of *troff* caused problems. I have had personal communication about another literate programming system that used a Macintosh WYSIWYG editor and the C language. All the documentation was done in German. David Ness created a CWEB-like system at *TV Guide*, which was not published or distributed.
- A NOWEB system that relaxed significant WEB requirements. Ramsey characterized his NOWEB as a "low-tech" literate programming system (Ramsey, 1991). It does not *indent* the source but passes it through.
- Jim Fox created c-~~web~~, which gives a nice listing of the source and some organization of the code. It can be argued that this is not a literate programming system because its indexing is minimal and the order of the code is dictated by C syntax. The c-~~web~~ package assumes that the C comments are written in TEX. The only other assumptions are that two C comments of a specific format appear near the start and end of the program, otherwise it is ordinary C with pretty output. The user simply TEX's the C source.

What is a Good WEB?

This is a question which still needs to be answered. It probably can't be answered today because there is not a large enough body of programs written in a literate style that are available for study. Later I will show one graph that appears to be a good indicator of characteristics of WEBs. I think several graphs of this type could be an effective indicator of quality.

The TEX system and the work reported in (Ramsey and Marceau, 1991) are the most significant examples from which we can begin to search for answers. Ramsey's work is based on a project in which there was not a single programmer (as in Knuth's work). More studies like this are needed.

Many of the characteristics I think of in using for evaluative purposes relate to the module. A module should be a 'sensible chunk.' In most cases, I interpret this to mean most modules should not be more than one screen of source, in spite of varying screen sizes of up to 40 lines. This is not an absolute rule, just a guideline. Some modules will be documentation only and may be several pages in length. In some cases, it is convenient to make each paragraph a module of its own. Further, there are modules that seem to be best presented when the documentation and code are **each** about a screen.

I can't yet answer the question as to what are the characteristics of a good web, but I think that studying with figures like Figure 1 and some statistics will begin to give us some 'normal characteristics.' These kinds of graphics make it easy to spot inconsistencies in documentation, modularity, and other causes of maintenance problems.

I do not doubt that there will be more than just one description of a good literate program. However, I think that most will have consistent "patterns" when analyzed in similar graphical manners using the "right statistics."

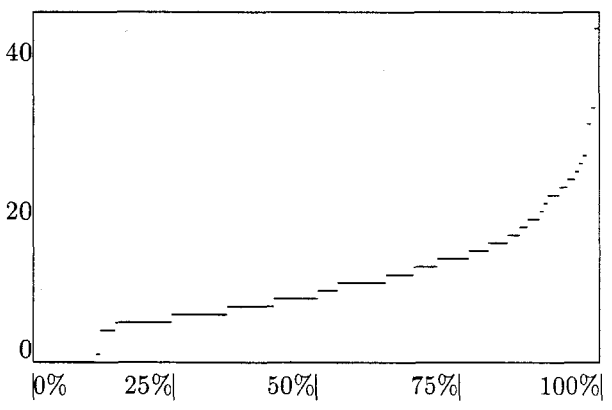


Figure 1. Percent modules *vs.* number of lines of code per module in `PS_Quasi.web`.

Mamoun Babiker wrote the tool to extract the data and create the TEX code for drawing the figure. Figures like this are better when they are done with a graphics package rather than relying on TEX's rules, for portability.

Some of the elements of a good essay will apply. Programming practices must change significantly for the spelling requirement to apply. Many codes and their documentation are full of acronyms and mnemonics. It can be argued that with today's systems and compilers accepting long variable names, there should be some change in this.

Tools for the Literate Programmer

I admit to believing in the value of literate programming even when I only had the usual editors and hardcopy (TEXed) output with the indices and cross-references. This was obviously **not** the preferred literate programming environment.

The editor is the most important "tool". It will be discussed briefly and most of the `web-mode` commands will be listed in a table. Other tools will be discussed without further introduction.

Editing environment. Mark Motl created the `web-mode` for GNU Emacs (Motl, 1990). It makes the editor sensitive to the rules of `WEB`, TEX, and some aspects of the high-level language. The table of contents, index, and index of modules that are helpful in studying `WEB`s are on-line. The user can select a variable while viewing the index and then with two keystrokes proceed to view each of the modules where the variable is referenced, in succession. All lists (module names, extra index entries like 'system dependencies', etc.) are kept on-line and the user only has to select from the list rather than retype them.

Details of `web-mode` are beyond the scope of this paper. A list of the functions that have been added to GNU Emacs to create a `web-mode` is shown in the appendix. We know it is used by a number of literate programmers on several types of UNIX systems and on MS-DOS machines and it certainly seems to be effective.

Counters. These are used for calculating the basic statistics and metrics that can be used to "measure" a web. We are using variants of these to build statistical summaries of `WEB`s, which we hope will lead to 'proven' statements describing the characteristics of good `WEB`s. The earlier figure was prepared by using one of these.

Table 1 was created by `wst` (a `WEB` statistics tool written by Mark Gaither) and is a count of the `WEB` commands in `tex.web`, while Table 2 is a count of the TEX commands. This data was extracted from version 3.14 of TEX which has 24,863 lines in its source and 523 of those lines have at least one `WEB` command. The complete Table 2 would be too long for this paper. However, the header is informative. I have left in only the eight most-used TEX control sequences. On the other hand, 69 TEX control sequences were used once and another 38 were used twice.

Table 1. WEB control sequences in tex.web.

Unique control sequences =	27
Occurrences of control sequences =	11609
Words in file =	127592
Unique control sequences/words =	0.0002
Control sequences/words =	0.0910

Control Seq.	Count
@	1325
@!	1655
@"	3
@#	28
@$	3
@&	5
@'	260
@*	55
@+	227
@,	8
@.	259
@/	685
@:	407
@;	90
@<	1774
@>	2868
@?	29
@@	16
@\	2
@^	259
@d	1216
@f	12
@p	178
@t	169
@{	5
@|	66
@}	5

Table 2. TEX control sequences in tex.web.

Unique control sequences =	202
Occurrences of control sequences =	6186
Words in file =	127592
Unique control sequences/words =	0.0016
Control sequences/words =	0.0485

Control Seq.	Count
\␣	381
\.	1728
\PASCAL	86
\TeX	477
\\	1431
\cr	125
\hang	166
\yskip	142

A recent scientific code of mine has 3,900 lines of WEB, 0.0072 unique control sequences per word, and 0.0431 control sequences per word — I think that this was expected. I consider this data to show an upper bound of the amount of TEX that might be included in a WEB. This is evidence that it is **not** necessary to be a TEXpert to use WEB.

Change file analyzer. Change files are analyzed in detail by this code (written by Mark Gaither). The UNIX `diff` utility is used to show exactly what changed in the change file for each module. Each module is flagged if it is only part of the lines of the module. The reason for this is that the GNU Emacs `web-mode` assumes that change files always contain complete modules. Also, since a module should be the minimum part of a code that can stand alone, it seems wrong to have only a part of it. If code is changed, documentation should also be changed.

A WEB structure viewer. `Web-view` (written by Kevin Borden) gives a high-level view of the structure of a WEB in a manner much like the browsing of the directory structure in a NeXT computer. This design was selected because the previous graphics representation of a WEB did not have sufficient screen to show enough of a module name to always be indicative of the purpose of the module. This tool is based upon X.

TeXbraces. This is a simple WEB (written by David Ness) that is used to flag those problem places where the dread braces get out of balance. I use this in the obvious fashion on both TEX and WEB sources. This functionality is present in `web-mode` and `tex-mode`.

Integration of RCS. Configuration management tools are a significant part of the toolbox on most code developers. RCS is a reasonably popular system that aids in keeping up with the changes from one version to the next. William Needels has recently finished a project on the integration of RCS and *imake* to automatically produce `Makefiles` for the generation of executables and documentation for systems created with WEBs. It shall be released shortly after packaging and another round of proofing.

The sources of all the tools that we support are available for anonymous `ftp` from: `ftp.cs.tamu.edu`. We are attempting to support all except the original environment of Marcus Brown. Another source for literate programming tools is `ftp.th-darmstadt.de`, the `pub/tex/src/webware` directory.

Community of Literate Programmers

Van Wyk indicated that only the creators of literate programming systems use them (Van Wyk, 1990). There is a significant international 'community of literate programmers.' I think the number of users is surprising since, to my knowledge, there has not been a conference or major portion of a conference dedicated to the exposition of the use of literate programming or the many unanswered questions on literate programming and environments.

It is difficult to accurately estimate the number of programmers using WEBs. I think the number is probably on the order of 1,000 or more because:

- Each semester several students contact me to show me some of their work or ask for help. These are not "my" students. They 'discover' WEB by scanning TEX directories.
- The distribution list for announcements about John Krommes' FWEB has several dozen addresses. I estimate that it does not include half the sites that really use it and most addresses probably represent several to many users.
- I know of three sites in Australia that are extensively using FWEB and which are not on the distribution list.
- A large multinational company is using CWEB as their methodology for code development.
- I have been politely chastised for not placing web-mode in the GNU or Archie archives. (It will be there soon.)
- There are at least two journals and four universities I know of listing literate programming as a viable/current research area. The subject literature contains about 100 papers, reports, and monographs from about 60 authors.
- I know of several professionals (such as David Ness) who are active literate programmers.
- Apparently a significant portion of the DANTE contributions to the TEX systems is being done in WEBs. I have seen the sources of BM2FONT and I know that Joachim Schrod and associates continue to use their CWEB.
- A discussion list for literate programming LitProg@SHSU.edu was started in July 1992 and generated many subscribers and more than 300k-bytes of messages in one month.

I intend to formalize the above list and seek permission to distribute the names and addresses of those I know. I believe that literate programming will be aided greatly by the publication of Don Knuth's new book, *Literate Programming* (Knuth, 1992).

Conclusions and Recommendations

I have no reservations in recommending the use of Knuth's original WEB, Levy's CWEB, or Krommes' FWEB. Our primary WEB system is Krommes' FWEB because it supports FORTRAN, FORTRAN90, RATFOR, C, and C++. We also find it to be the best-documented of the WEBs. It was created by extending Levy's CWEB.

I believe that the use of web-mode has made literate programming easier to introduce to my students. A similar tool (but not as complete) should soon be available for VMS systems based upon LSEDIT.

Ramsey's Spider system is a good tool for creating new WEBs and his NOWEB is certainly worthy of study.

Future Work

The questions related to determining the quality of a literate program are still not answered. I believe that we are beginning to have tools and examples that will help answer these questions or at least give us some general ideas. We need a significant body of WEBs written by teams and a range of programmers for study.

I believe that we need literate programming systems whose output can be tailored to personal tastes. For example, a Pascal code could have <= instead of $\leq$, braces instead of **begin-end**, and physical things like page size, etc.

Generic WEBs should be available for other styles of languages such as VAX DCL files, UNIX scripts, hand-held calculator code, etc. Norman Ramsey's noweb may be a good vehicle for this since it does not reformat the code. One of the most underused features of FWEB is that it allows the creation of literate style files for TEX macro writing. This is an obvious improvement over the .doc to .sty contribution in Leslie Lamport's original LATEX.

Much of today's computing is no longer considered to be a single-language environment. Scientific computing is often a mix of high-level languages. FWEB allows the mixture of FORTRAN, C, etc. The inclusion of MATLAB, GNUPLOT, etc., should be included in a literate form.

A comment

In the interest of brevity I have omitted many references that should be a part of the complete paper. I point to Nelson Beebe's bibliography on literate programming which is available by anonymous ftp from science.utah.edu for a complete list of many relevant citations.

Acknowledgements

TUG's anonymous reviewer made **many** comments that contributed to this paper. The thoroughness of that review is appreciated.

David Ness and I have had many conversations about literate programming and his vision has certainly helped my views and understanding about literate programming and several other aspects of the computing professions.

References

Brown, Marcus E. "An Interactive Environment for Literate Programming". Ph.D. thesis, Texas A&M University, College Station, TX, August 1988.

Brown, Marcus E. and Bart Childs. "An interactive environment for literate programming". *Journal of Structured Programming*, 11(1), pages 11–25, 1990.

Fox, Jim. "Webless literate programming". *TUGboat*, 11(4), pages 511–513, November 1990. `u.washington.edu`

Gragert, Peter, and Marcel Roelofs. *Reduce* WEB *version 3.4.* `utmfu0.math.utwente.nl`

Gragert, Peter. Personal communication.

Guntermann, Klaus, and Joachim Schrod. "WEB adapted to C". *TUGboat*, 7(3), pages 134–137, October 1986. *This* WEB *is no longer supported. They recommend the Levy/Knuth* CWEB. `schrod@iti.informatik.th-darmstadt.de`

Knuth, Donald E. "The WEB system of structured documentation". Stanford Computer Science Report CS980, Stanford University, Stanford, CA, September, 1983. `labrea.stanford.edu`

Knuth, Donald E. "Literate programming". *The Computer Journal*, 27(2), pages 97–111, May 1984. Also appears as chapter 4 in "*Literate Programming*".

Knuth, Donald E. *TEX: The Program*, volume B of *Computers & Typesetting*. Reading, MA: Addison-Wesley, 1986. ISBN 0-201-13437-3.

Knuth, Donald E. METAFONT: *The Program*, volume D of *Computers & Typesetting*. Reading, MA: Addison-Wesley, 1986. ISBN 0-201-13438-1.

Knuth, Donald E. "Remarks to celebrate the publication of *Computers & Typesetting*" *TUGboat*, 7(2), pages 95–98, June 1986.

Knuth, Donald E. *Literate Programming*. Center for the Study of Language and Information, Stanford University, CA: Distributed by Univ. of Chicago Press, ISBN 0-937073-80-6, 1992.

Krommes, John A. *The* FWEB *System*. Princeton University. 1989. `lyman.pppl.princeton.edu`

Levy, Silvio. "WEB adapted to C, another approach". *TUGboat*, 8(1), pages 12–13, April 1987. `princeton.edu` and `labrea.stanford.edu`

Motl, Mark B. "A Literate Programming Environment Based on an Extensible Editor". Ph.D. thesis, Texas A&M University, College Station, TX, December, 1990. `csseq.cs.tamu.edu`

Ramsey, Norman. "Weaving a language-independent WEB". *Communications of the ACM*, 32(9), pages 1051–1055, September 1989. `princeton.edu`

Ramsey, Norman , and Carla Marceau. "Literate programming on a team project". *Software—Practice & Experience*, 21(7), pages 677–683, July 1991.

Ramsey, Norman. "Literate programming tools need not be complex". Submitted for publication, 1991.

Sewell, E. Wayne. "How to MANGLE your software: the WEB system for Modula-2". *TUGboat*, 8(2), pages 118–122, July 1987.

Sewell, E. Wayne. *Weaving a Program: Literate Programming in* WEB. New York, NY: Van Nostrand Reinhold, 1989. ISBN 0-442-31946-0.

Thimbleby, Harold. "Experiences of 'literate programming' using cweb (a variant of Knuth's WEB)". *The Computer Journal*, 29(3), pages 201–211, June 1986.

Van Wyk, Christopher J. "Literate Programming: Moderator's Introduction". *Communications of the ACM*, 32(6), page 740, June 1989.

Van Wyk, Christopher J. "Literate programming—an assessment". *Communications of the ACM*, 33(3), pages 361 and 365, March 1990.

Appendix

Listing of WEB-MODE Commands by Functionality

Functionality	Command	Key Binding
Movement Among Buffers (Files)	web-goto-buffer-by-name	C-c b n
	web-goto-buffer-change-file	C-c b c
	web-goto-buffer-include-file	C-c b i
	web-goto-buffer-web-file	C-c b w
Movement Among Modules	web-goto-module	C-c g m
	web-next-module	C-c n m
	web-previous-module	C-c p m
Interactive Access to and Movement Among Sections	web-goto-section	C-c g s
	web-next-section	C-c n s
	web-previous-section	C-c p s
	web-view-section-names-list	C-c v s
Interactive Access to Index	web-next-index	C-c n i
	web-previous-index	C-c p i
	web-view-index	C-c v i
Interactive Access to Modules	web-next-define	C-c n d
	web-next-use	C-c n u
	web-previous-define	C-c p d
	web-previous-use	C-c p u
	web-view-module-names-list	C-c v m
Change File Editing and Movement	web-edit-module	C-c e m
	web-goto-change-corresponding-to-module	C-c g c
	web-next-change	C-c n c
	web-previous-change	C-c p c
Web Structure Information	web-delimiter-match-check	C-c d m
	web-determine-characteristics	C-c d c
	web-view-changed-modules-list	C-c v c
	web-what-change	C-c w c
	web-what-module	C-c w m
	web-what-section	C-c w s
Miscellaneous	web-insert-index-entry	C-c i i
	web-mode-save-buffers-kill-emacs	C-x C-c
	web-rename-module	C-c r m

A High-Performance TEX for the Motorola 68000 Processor Family

Steve Hampson and Barry Smith
Blue Sky Research
534 SW Third Avenue
Portland, OR 97204
Phone: (800) 622-8398
Internet: barry@reed.edu

Abstract

TEX is a large, computationally-intensive program that, thanks to its author, is extremely stable and thoroughly debugged. We describe herein some aspects of our recent work on an assembly language implementation of TEX for the Motorola 68000 processor family. Particular attention is given to memory reference and procedure call templates, performance measurement tools and techniques, and stupid compiler tricks. We also compare the results of our work with those of more conventional methods.

Background

Each of the authors has a good deal of practical experience with Pascal compiler engineering. We were each previously employed in senior technical positions at Oregon Software, a small company specializing in high-performance compilers. (Curiously, our times at Oregon Software did not overlap, and we did not meet until later.)

TEX, of course, is a large computer program written (by D. E. Knuth) in WEB, a structured design language that can be processed to yield a (more or less) standard Pascal program. The TEX program is very stable and thoroughly debugged, thanks to its author and the many users involved in its development.

Motive

Blue Sky Research is the publisher of *Textures*, a TEX-based desktop publishing system for the Apple Macintosh family of computers. The interactive orientation of the Macintosh compels its programmers to be attentive to human-scale performance, and we found the speed of TEX itself to be a limiting factor in making Textures more interactive. (Also, frankly, as compiler writers, we became unhappy whenever we looked at the code produced by the commercially-available compilers.) We, of course, were aware of the traditional disadvantages of assembly code, but thought that the extreme stability of the TEX program made this a special case.

Procedure

We did not propose to recode TEX directly in assembly code, being aware of the programmer's maxim "80% of the time is spent in 20% of the code." (Actual measurements suggest that, for TEX, 95% of the time is taken by less than 3% of the instructions!) We instead proposed (and carried out) a course of action roughly as follows:

(1) Produce a special-purpose Pascal compiler to generate readable basic assembly code;
(2) tune the code generation of the compiler based on extensive measurement and examination of critical code sequences;
(3) when no further compiler improvements appear worthwhile, throw the compiler away and continue with the compiler-generated assembly code as a base;
(4) identify critical routines and segments and rewrite "by hand"; and
(5) repeat step 4 until it no longer appears worthwhile.

(Step 3 was referred to within our team as "making the jump to hyperspace." Steps 4–5 are continuing as of this writing.)

Examples

This portion of the paper assumes some familiarity on the part of the reader with the flavor of assembly language and the structure of TEX itself. For readers not familiar with the Motorola 68000 family architecture, it is by and large a 32-bit two-address multiple-word instruction set with 8 general-purpose

arithmetic registers (D0 – D7) and 8 32-bit address base registers (A0 – A7). A7 is conventionally used as the stack pointer (SP).

Example 1: Access to the "mem" array. The mem array is TEX's principal memory structure, and is referenced by many parts of the program. Since Textures incorporates a "large" TEX, each element of mem occupies 8 bytes; furthermore, since in Textures the mem array can grow as needed, it is referenced by a pointer that may change value when the array changes size.

The straightforward 68000 instruction sequence to access item "n" is as follows, assuming "n" is in D0:

```
asl.l  \#3,D0          ; shift N left 3 bits
move.l MEMPTR,A0       ; load address reg
move.l 0(A0,D0:L),D1 ; mem[N].rh -> D1
```

This is a relatively expensive sequence, considering its wide-spread use. Especially costly is the instruction to load the base address of mem, requiring 4 bytes for the instruction itself and also a 4-byte memory fetch.

On the 68000, address register A6 is conventionally used as a procedure frame pointer, maintaining the stack offset on procedure entry for reference to parameters and local variables. (This simplifies code generation within the procedure as the stack grows and shrinks.) As it happened, the Oregon Software Pascal – 2 compiler we used as a base for our efforts was originally designed for the Digital PDP – 11 computer, which has no such frame pointer. The compiler therefore already was capable of computing stack offsets without the benefit of the frame pointer, freeing register A6 for other uses. We dedicated it to serve as the pointer to mem, and modified the compiler to generate special code for references to that variable.

We then went through the code to TEX, identifying each variable and expression that could serve as a mem reference. The compiler pre-shifted constants in mem indices, and we changed each computed expression to pre-calculate the left shift (equivalent to multiplying by 8). (Most pointers are produced by arithmetic on pointers, so we changed statements like "p:=p+1" to "p:=p+8".) The resulting code eliminated the need for the shift instruction, so the code above simplifies to a single instruction:

```
move.l 0(A6,D0:L),D1 ;mem[N].rh -> D1
```

(This was *much* easier to write about than to actually perform; before we could be satisfied that we had identified every reference to mem, we found it necessary to create tools that would give us a full trace of each mem reference to compare with a standard trace.)

Example 2: instruction counting. We used several TEX jobs as performance benchmarks. The largest and most "life-like" of these was *The TEXbook*, which produces 494 typeset pages. The performance analyzer provided with the Macintosh development system is an interrupt-based profiler that samples the program counter every few milliseconds and produces an execution time profile by procedure blocks. While this was very indicative of major program flow, we found it somewhat unstable and too coarse for critical sections. We then built an instruction-by-instruction counter that produced an assembly listing with execution counts attached to each instruction. Here is a partial summary of counts for *The TEXbook* from April 20:

calls	instructions	%/total	procedure
5 703 564	190 236 375	16.435	get_next
1	152 982 983	13.217	main_control
82 008	60 914 195	5.263	hpack
53 313	57 517 172	4.969	hlist_out
229 313	55 271 629	4.775	try_break
8 036	46 325 505	4.002	line_break

This tool was somewhat expensive to use, slowing execution by a factor of 150 or so, but the information was happily consistent and precise. (Perhaps seductively so; we sometimes needed to remind ourselves that instruction counts were *not* directly related to processing time. In more than one case we chose instruction sequences with more instructions but faster execution, according to processor timing calculations.)

Our measurement tools gave us more than enough information to identify critical sequences for hand work. (a target-rich environment for assembler jockeys, so to speak.) Here is a similar summary from May 23 (the calls are identical):

instructions	%/total	procedure
128 071 928	13.322	get_next
109 146 891	11.353	main_control
53 136 456	5.527	try_break
51 951 382	5.404	hlist_out
46 203 165	4.806	line_break
43 949 702	4.572	hpack

Example 3: Procedure calling sequences. A slightly different view of instruction count data summarizes procedures in order of the number of calls:

calls	instructions	inst/call	procedure
5 703 564	190 236 375	33.4	get_next
2 840 411	32 782 196	11.5	get_xtoken
1 998 121	12 033 901	6.0	get_avail
1 150 392	39 422 420	34.3	get_node
1 150 334	14 954 342	13.0	free_node
1 057 213	11 336 157	10.7	get_token

Here, for example, we can see that we should consider replacing high-frequency calls to "get_avail" with the equivalent in-line code. (In fact, Knuth has already done this with a "fast_get_avail" WEB macro; we carried this idea a little further in our assembly code.)

More importantly, the sheer number of calls to these routines focused our attention on calling sequence overhead, both at the compiler template level, and later on special-case sequences for certain routines. Each single instruction eliminated from the calling sequence for "get_next" reduced the run-time for *The TEXbook* by almost 1 second on our Quadra test platform.

Perhaps the most interesting lesson from our experiments was the (in-)validation of some of the design concepts of the Pascal-2 compiler. The design team made some assumptions about program structure that are not true for TEX, e.g., that the procedure structure corresponded to significant work elements of the problem. We found instead that, in TEX, a typical procedure has a relatively small critical path that corresponds to the large majority of uses, and a much larger body of code for special cases and error recovery. Unfortunately, the compiler generated significant amounts of register traffic on each entry to "optimize" register usage over code that was never used!

Conclusions

The overall performance of TEX has improved by roughly a factor of three as of this writing, compared to previous Macintosh implementations via a standard (Apple) Pascal compiler. We believe there are significant gains still to be realized, although we are clearly seeing a diminishing return on efforts. So far, the reliability and maintainability are more than satisfactory; the assembly language TEX described herein is shipping in our current version of Textures, with no TEX problems reported to date.

Thanks

We would like to express our thanks to Professor Knuth for the TEX program itself, and especially for the TRIP test program, which has allowed us to easily locate bugs that would have been vanishingly obscure in more "normal" uses of TEX.

Using a High-Level Language as an Aid in Writing TEX Documents

Harry L. Baldwin, Jr.
San Diego City College
1313 12th Ave.
San Diego, CA 92101

Abstract

Certain mathematical procedures associated with the preparation of a document are best handled by some high-level language other than TEX. One example is the generation of random numbers and the subsequent ordering of those numbers as a means of scrambling lines or groups of lines in the preparation of different forms of a test. Other examples involve the generation of curves and angles for inclusion in graphs and diagrams. An alliance of the True BASIC programming language and TEX has proven to be a useful combination for the efficient creation of TEX source files.

Introduction

Although TEX was designed for quality typesetting, and to reach that goal Don Knuth filled his program with many capabilities, most users would probably admit that number crunching is more easily done in some other high-level language. In my four years of using TEX as a mathematics teacher, I have found several applications in which my attempts to produce high-quality output are simplified by using TEX in close association with another language. This article will be a chronological account of some of those applications.

The high level language that I use is not one of the latest exotic creations to which *Byte Magazine* might devote an entire issue, but rather what has lately been looked upon as the Rodney Dangerfield of computer languages, BASIC. My first meshing of TEX and BASIC came when I implemented PCTEX and needed an editor capable of creating an ASCII file. Although I was doing True BASIC number crunching on the PC, all my word processing was being done on a NorthStar Horizon computer (anybody remember that old warhorse?). I decided to write my TEX source files using the True BASIC editor until something better came along.

The True BASIC editor has been working so well that I've never bothered searching for a better one. The only time it complains is when I occasionally accidently hit F9 (the "RUN" command) at which time it politely tells me that "`\documentstyle{book}`" is an illegal statement. All the editing features I need are available, such as search and replace, block move, copy, delete, and so forth, together with the capability to insert another file into the working file at any desired location.

Scrambling Questions and Answers

My first uses of TEX were handouts and tests (where "tests" means both long "examinations" and short "quizzes"). Since the number of students in my classes kept getting larger each semester, but the classroom sizes somehow stayed the same, roving eyes during tests became a problem. Giving different versions of a test seemed to be the appropriate action to take. The versions would differ from one another by either scrambling the order of choices to each multiple-choice question, or by scrambling the order of the questions on the test, or perhaps by using both of these techniques. Rather than have TEX create the different versions of a test — an approach adopted by Don De Smet (1991) — my strategy is to let True BASIC do the scrambling: A TEX source document (let's assume it is named `TEST.SCR`) is input to a BASIC program (called `SCRAMBLE.TRU`) that searches the lines for certain code characters. The codes identify lines whose positions will be changed in a manner that will be described below. A new source file (let's name it `TEST.TEX`) is written to the hard disk, all ready to be compiled under LATEX.

Two types of scrambling can be done: scrambling of questions and scrambling of answers. A multiple-choice question comprises a group of lines that contain the question's "root" and the five "answers" (the single correct answer and the four "distractors"). Scrambling of questions requires changing the positions of blocks of lines, while scrambling of answers involves shuffling a few contiguous lines within the block. The type of scrambling that will be done is determined by "code characters" that appear on some lines of `TEST.SCR`. The code characters are placed to the far right end of the line for easy

identification during screen editing of the source file. The leftmost character in any sequence of code characters is the comment character %, which will not interfere with the subsequent compilation of the document.

The beginning line and ending line of a question that is to be scrambled are identified by the code characters %B and %E in character positions 76-77. These lines, together with all lines in between, form a block whose position will be scrambled. The end of the last question in a group of questions that is to be scrambled is identified by %EL in positions 76–78.

A question that is to be scrambled need not be a multiple-choice question, but if it is, usually we want the answers to be scrambled also. Each of the answers in a multiple-choice question (we will assume there are five) will be an argument of a macro designed to output the answers appropriately. In TEST.SCR the answers must be written on five contiguous lines, one answer per line. (If an answer is too long to fit on one line, then that answer should be made into a macro, and the macro command put on the line.) To identify an answer that is to be scrambled, the comment symbol % is placed in character position 78 on that line. For example, if only the first four answers are to be scrambled (maybe the fifth answer is "all of the above") then four contiguous lines would contain a percent symbol in position 78.

The scrambling is done as follows: The source file TEST.SCR is read into memory, each line destined to become an element of a one-dimensional string matrix that we'll name TEST$. As each line is entered, it is examined for a percent symbol in position 78, which would identify that line as an answer to be scrambled. If such a symbol is not found, then the line is merely appended to TEST$, but any sequence of answer lines that are to be scrambled are first stored in another temporary array. Pseudo-random numbers are generated (the "pseudo" is for the purists — I'll just call them random numbers) and placed in another column of that array. The rows of the array are then reordered so that the random numbers increase as we go down the column, and the lines associated with those numbers have their positions changed as well. The reordered answer lines are then transferred to TEST$. This process continues until all lines of TEST.SCR have been entered.

The lines of the matrix TEST$ can now be output to TEST.TEX, and the only change from TEST.SCR will be the order of answers. However, if scrambling of questions also is desired, then some more juggling is required: As each line of TEST$ is output, it is examined for %B in positions 76–77, which would mark the "begin-line" of a question whose position is to be scrambled. If not found, then the line is merely output to TEST.TEX; if found, output is suspended while the program searches for the corresponding "end-line" of that question, identified by %E is positions 76–77. The line numbers that identify that question are stored in a row of a "line-number matrix", and search continues for another begin-end pair of line numbers, which will be stored in the next row of that matrix. After the end-line of the last question to be scrambled is identified (by %EL in positions 76–78), then a random number is assigned to each row of the line-number matrix, and those rows are reordered. The output of TEST$ is then resumed, with the order of the lines indicated by information contained in the rows of the reordered line-number matrix.

Sometimes a test should not have all questions scrambled together, but rather, say, the first twenty easier questions scrambled, and then the next thirty more-difficult questions scrambled. This can be done by the above scheme, by merely terminating each group that is to be scrambled with a line containing %EL in positions 76-78.

Each time a True BASIC program is run, the same sequence of random numbers is generated. To obtain a version of a test, SCRAMBLE.TRU asks for the date and the form number, and then discards the first n random numbers, where the number n is given by $n = 366 \cdot (\text{yr}+\text{form}) + 31 \cdot \text{mo} + \text{day}$. For example, to generate the fifth form of my last April Fool's test I entered the "seed" 0401925, and the first 35 627 random numbers were discarded (slowing the execution by less than a second). If, for some reason, I need to recreate the version I can input the same seed and obtain exactly the same test.

The macro chosen to output the answers to a multiple-choice question depends on the lengths of those answers. Five macros have accommodated all possibilities I've needed so far; letting $r{:}s{:}t\ldots$ represent r answers on the first line, followed by s answers on the next line, etc., the macros will print the answers following one of these patterns: 5, 3:2, 2:2:1, 4:1, and 1:1:1:1:1.

An example of what a couple of questions in the source file might look like, and what output might be produced, is shown in Figure 1 (in the Appendix). The command \QQQQR is the macro that outputs four answers on one line and the fifth answer on the second line (such an arrangement would only be used if the fifth answer is not to be included in the scrambling); the command \QQRRS outputs two answers on the first line, two on the second, and one

on the third; (can you guess what \QRSTU would output?). These macros begin by incrementing the question number. Then (in a \vtop, so that a question won't be split by a pagebreak) a horizontal rule is drawn for question separation, followed by printing the root of the question (the first argument of the macro) and then the answers (arguments two through six). The macro concludes with another horizontal rule.

Generating a Test

SCRAMBLE.TRU worked so well that when the City College Mathematics Department decided to implement a departmental final exam for the Beginning Algebra course (and I was drafted to create this final), it seemed reasonable to let True BASIC compose the entire test. The test-making philosophy our department adopted was this: the final exam would be formed by selecting questions from a test bank that is so large that test security is of no concern. Currently our test bank contains 215 "master questions" (we are aiming for 300), each master question having ten quite-similar "versions". Since a final exam would be formed by selecting 60 master questions from the bank of 215, *and* any of the ten versions of each question will be randomly selected, *and* the 60 test questions would be scrambled (in two groups), *and* the answers would be scrambled, we feel that any student capable of beating the system by remembering all the questions and their correct answers is probably bright enough to realize that it would be easier just to learn the algebra.

Each master question (comprising ten versions) is stored as a separate file; for example, Q130 is the file that contains master question 130. If that master question is selected for a test, then part of that file (a version of the question) is destined to be input by the BASIC program MAKETEST.TRU that will be composing the test. A random number determines which of the ten versions will be selected. Each version occupies 15 lines of the file, so version 4, for example, would extend from lines 46 through 60, with perhaps the last few lines being blank.

Refer to Figure 2 (in the Appendix) as we look at what MAKETEST.TRU does with the 15 lines of a question version that has been randomly selected. The first line contains five pieces of information about the question, with each of these pieces of information appearing in a specific location on the line:

Version number. A whole number, in character positions 9-10, identifies the version selected. During any subsequent scrambling, MAKETEST.TRU will keep track of the question master number (one of the 215 questions in the test bank) and the version number (1 through 10). After the test is constructed an answer key is printed that will include this information. If a bug in the question shows up then it is easy to locate the offending version.

Scramble code. A whole number, in character position 21, tells how many answers are to be scrambled: 0 (for no scrambling), or 3 (the first three answers), or 4, or 5.

Answerline code. A whole number (1, 2, 3, 4, or 5), in character position 36, tells MAKETEST.TRU what macro to use when outputting the answers. For example, code number 1 is five answers on one line.

Rootlines code. A whole number, in character position 49, tells how many lines contain the root of the question. The root lines immediately follow this first codeline, are written in TEX, and will be inserted directly into the TEX document being constructed. The maximum number of root lines is 9; although we have never needed that many lines, a macro could be defined and placed in the preamble if necessary.

Correct answer. A letter (A, B, C, D, or E), in character position 59, identifies the correct answer. The five answers are on the five lines that follow the root line, with exactly one answer per line. Before scrambling, answer A is the first answer listed, B is the second, and so forth. When writing a test question, I usually work the problem and write the correct answer on the first answer line, and then play the part of an inattentive student as I make up the distractors. During the subsequent scrambling of the answers, MAKETEST.TRU tracks the correct answer for inclusion on the answer key.

The construction of a test by MAKETEST.TRU proceeds as follows: The person creating the test selects the master numbers of the questions that will be on the test. These numbers are written in a file, perhaps named FIN-S92 if they are the questions to be used on the final exam for the Spring semester in 1992. MAKETEST.TRU is then run, and begins by asking for the test title, subtitle, and special instructions. (Provision is made for placing this information in a file before running MAKETEST.TRU, and merely entering the name of the file when prompted.) MAKETEST.TRU next asks for the number of questions, for a random-number seed, and for information on what groups of questions are to be scrambled. (If no question scrambling is to be

done, then the questions will be output in the same order as they appear in the question file.) Then the master-question numbers are scrambled, and versions randomly selected.

The output file (let's again call it TEST.TEX) is now ready to be created. MAKETEST.TRU first calls a file named PREAMBLE.TEX that contains all the commands in a LATEX document preamble. These lines are output to TEST.TEX. Then the title information and instructions are appended. Finally, the randomly selected version of each master question is input: the master number of a question is used to form the name of the file that holds the ten versions, the lines of that file are entered into memory, and the 15 lines that correspond to the selected version are retained while all other lines are discarded. Information is extracted from the codeline (the first line): how many of the following lines contain the root, how many answers are to be scrambled, how many answers should appear on each line, and which is the correct answer. The root of the question is then inserted as an argument of a macro written by MAKETEST.TRU using BASIC string functions, the answers are scrambled and inserted as arguments into the appropriate macro (also written by MAKETEST.TRU), and all of the lines are written to TEST.TEX. This procedure is repeated until the lines that will print all questions have been output to TEST.TEX.

The answer key is then constructed. After a \pagebreak, four columns of information are presented: the question number as it appears on the test, the corresponding master-question number, the version number, and the answer. The last hurrah of MAKETEST.TRU is to append \end{document}.

After the file containing the master question numbers has been created, the running of MAKETEST.TRU takes only a few minutes. Even though a lot of computation and scrambling is involved, on an 80386 running at 33 megahertz the questions are written to TEST.TEX at about one per second. For the final exam in Beginning Algebra this last Spring semester, all sections were to be given a test that used the same master questions. Since the tests were to be given at different times, eight runs were made using different seeds to obtain eight different forms, each form containing eight pages of questions plus an answer key. Creation of all eight forms of TEST.TEX, compiling under LATEX, and then obtaining HP LaserJet II output ready for photocopying took about 40 minutes.

Writing the ten versions of a master question goes quickly. Since all versions are to be *very* similar, the roots of the versions usually will be the same, except perhaps for a mathematical expression or a few words. The codelines for each version usually differ only in the version number. Therefore, one version can be written, duplicated nine times, and then changes made to those parts where the versions differ.

Another BASIC program, PRT-VER.TRU, was written that will read in a master file and print all versions — a great help in proofreading the questions. Still another program, PRT-ALL.TRU, prints a single version from each master question. The output from PRT-ALL.TRU is what instructors look at when selecting master questions for a test, and what students look at when they are curious as to what might appear on a test.

This testing strategy can be a morale raiser for students, since they needn't fear being hit with a question of a type they have never seen before. But since the students would see only one version from each master question (printing all ten versions would make a booklet of several hundred pages), great care must be taken to insure that all versions are of the same difficulty. For example, a question that asks for one of the binomial factors of $x^2 + 7x + 12$ is not considered to be of the same difficulty as asking for one of the binomial factors of $x^2 - 4x - 12$, for this second trinomial involves both positive and negative integers. To insure that much thought is applied to the construction of the questions, a verbal description of the limitations is written for each master question. For example, the description of one trinomial-factoring question is as follows:

> A given second-degree trinomial has two first-degree binomial factors, one of which is to be selected from a list of five possibilities. The coefficient of the second-degree term is 1; the constant terms of the binomials will be non-zero integers, of different sign, having magnitudes less than 10.

Of course, a student who only knows his multiplication table up through fives might not consider the factoring of $x^2 + 3x - 54$ to be of the same difficulty as $x^2 + 3x - 10$, but at least we tried.

Figure 2 (in the Appendix) shows an example of a version from each of two master questions, as they would appear in the master-question files. Also shown is the output of these same questions as produced by MAKETEST.TRU.

Graphics Output

Other opportunities to mesh True BASIC and TEX arose from my efforts to produce mathematically-accurate drawings for my handouts, and for a geometry book recently completed (Baldwin, 1992).

A brief description will be given here of only two of several BASIC programs that were helpful.

Most of the graphs I construct for tests and handouts require only a simple xy coordinate system framework, on which an arbitrary curve will be drawn. In the TEX document, a macro `\XYgrid` will construct the framework, label and place scales on the axes, and then leave the "cursor" at the left end of the top gridline of the framework. Another macro enters the LATEX picture environment, places the origin at the intersection of the x and y axes, and sets `\unitlength` equal to the unit distance on the framework. The TEX document is then saved while BASIC constructs the curve.

A True BASIC program named `CURVE.TRU` will write LATEX `\multiput` statements to a temporary file, which will then be inserted into the TEX document at the proper place. Each `\multiput` statement will place dots of a selected size along a straight-line segment defined by the endpoints. By making the segments quite short, and linking them together, a curve can be drawn.

The curve is defined by functions of a parameter — x and y each as a function of t — which are placed very near the beginning of `CURVE.TRU`. These functions are defined by editing `CURVE.TRU` before running (which is much easier that having `CURVE.TRU` ask for the functions during running). When run, `CURVE.TRU` first asks for the framework scale, the starting and ending values of the parameter t, and "delta t" (the change in t, which will influence the lengths of the straight-line segments. Finally, `CURVE.TRU` asks for the size and spacing of the dots that will form the curve. After a couple of seconds the program announces that it is finished, having written a series of `\multiput` commands to a file named `TEMP.TRU`. The TEX document is then reloaded, and the lines in `TEMP.TRU` are inserted at the proper location.

A dot diameter of 1 point (obtained by using the smallest possible LATEX `\circle*`) with a spacing between centers of 0.7 points creates a curve whose thickness matches a LATEX `\thicklines` line almost perfectly. Thinner curves can be made by using periods in smaller fonts.

Several variations of `CURVE.TRU` have been written: dotted curves, dashed curves, curves where checking is done so plotting is restricted to a certain area, and some other variations. Rather than have one giant elegant program that will do everything, I have found it more efficient to select one of a number of special programs available, so that a lot of queries needn't be answered during running. For example, `CURVE-DU.TRU` ("dashes, unlimited") makes a dashed curve, without checking for out-of-bounds points, while `CURVE-DL.TRU` ("dashes, limited") asks for the limits on x and y.

Some curves result in *many* `\multiput` commands being inserted into the TEX file — perhaps a few hundred. Occasionally I get a "`Sorry, TeX capacity exceeded`" message. Since I don't understand any of the rest of the message, I usually just shrink the curve or increase the delta t and try again.

I often draw curves using the excellent curve-drawing capabilities of PICTEX, especially circles and ellipses. For other curves, PICTEX requires computing the coordinates of some points that lie on the curve and either entering these coordinates into the TEX document or storing them on a file. Rather than worry if enough points have been computed, or if something strange is happening on the curve between those points, I just use `CURVE.TRU`. The compiling times for TEX documents that produce curves by the two methods are about the same.

Figure 3, immediately below this paragraph, shows an example. The framework (axes and labels, grid lines, and the scale) was generated by `XYgrid` (a TEX macro), but the curve was formed from 72 `\multiput` commands generated by `CURVE.TRU`, and inserted into the TEX source file for the document you are reading.

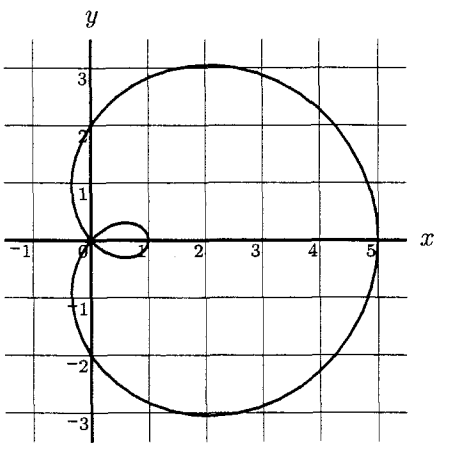


Figure 3: limaçon: $r = 2 + 3\cos\theta$

If an angle in a drawing is accurate to within about one degree of its labeled measure, then it takes a highly-trained eye to detect the error. Whenever possible, I use the line-drawing capabilities of the `picture` environment of LATEX to draw an angle. The `\line` command in LATEX can draw a vertical line, or a line having slope $\pm y/x$, where x and y are whole numbers six or less. The available slopes allow an angle of any measure to be constructed with

an error less than one degree, although not with an arbitrary orientation. If a figure has two or more angles, or if the orientations of one or more angles are constrained, then LaTeX `\line` commands may not create a sufficiently-accurate drawing, so the sides of those angles may have to be drawn by another method. Furthermore, pictures of angles often require an arc to indicate the measure of the angle, and that arc might terminate with an arrow at one or both ends. Sometimes a picture should show the sides of an angle to be rays (an arrow pointing away from the vertex); sometimes an angle's sides are segments. A right angle might be indicated by a square placed at the vertex. All of these facts served as the incentive to let True BASIC do the job, by means of a program named `ANGLEARC.TRU`.

A LaTeX picture environment is established, with the scale and the origin chosen, and the TEX source file is saved. When `ANGLEARC.TRU` is run, it asks (and I answer) these questions:

What is the scale?

What are the coordinates of the vertex?

What is the direction of the angle's side?

What is the length of angle's side?

Is the side a ray? If yes, a LaTeX `\thicklines` vector of zero length having the closest-possible direction as the side will be placed at the end of the segment that forms the side.

Should a right-angle square be drawn at the vertex? If yes, then what is the size of the square?

What is the radius of the arc that will indicate the angle's measure? (An answer of zero indicates no arc is to be drawn, and the next two questions are skipped.)

Should arrows be placed at the terminal end of the arc, or at both ends of the arc? If yes, LaTeX `\thinlines` vectors of zero length (and closest direction) are placed at the appropriate endpoints of the arc.

What is the central angle of the arc?

Is another side to be drawn? To construct the terminal side, answer yes and the questioning begins anew.

As the questions are answered, `ANGLEARC.TRU` constructs the proper LaTeX `\multiput` and `\put` commands, and outputs them to `TEMP.TRU`. After no more angle sides are to be drawn, control returns to the BASIC editor and the TEX source file is reloaded. The cursor is moved to the proper location, and all the lines of `TEMP.TRU` are copied into the source file.

An example of output produced mainly by commands generated by `ANGLEARC.TRU` is shown in Figure 4, immediately below this paragraph. Except for the two angle labels and the Figure label, all of the drawing was done by commands generated by `ANGLEARC.TRU`. The vector arrows, and the dots that form the arcs, were placed with `\put` commands. The angle sides were formed from `\multiput` commands, which placed 1-point dots with their centers spaced 0.7 points apart. The right-angle square was formed similarly, but using smaller dots. The three labels were placed "manually" after returning to TEX.

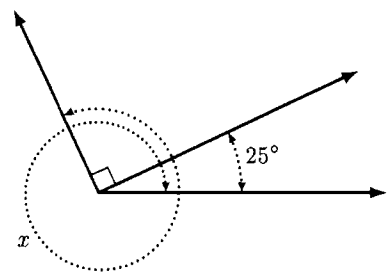


Figure 4: $x = ?$

Much use was made of `\ANGLEARC.TRU` and other True BASIC programs during the writing of my geometry book. Figure 5 (in the Appendix) shows part of a page from the book, one of many in which this program played an important role.

Every program described in this article could, of course, be implemented in other high-level languages — perhaps even in TEX. But for ease and convenience of use, True BASIC has earned my respect. I've enjoyed being a witness to the wedding of True BASIC and TEX, and I'm sure that the marriage will be a long and happy one in my computer system.

Bibliography

Kemeny, John G., and Thomas E. Kurtz. *True BASIC Reference Manual.* West Lebanon, N.H.: True BASIC, Inc., 1990.

De Smet, Don. "TEX Macros for Producing Multiple-Choice Tests." *TUGboat* 12(2), pages 261–268, 1991.

Lamport, Leslie. LaTeX *User's Guide and Reference Manual.* Reading, Mass.: Addison-Wesley, 1986.

Wichura, Michael J. *The* PICTEX *Manuel.* TEXniques 6, 1987.

Baldwin, Harry L. Jr. *Essential Geometry.* San Francisco, Calif.: McGraw-Hill, 1992.

Appendix

```
character                                                          character
 position                                                           position
   1                                                                  77
   ↓                                                                  ↓
   \midexamplespacer      % Draws a horizontal rule
   %13                                                               %B
   \QQQQR{Which of the four complex numbers listed below has the
       greatest modulus?}
   {$5+5i$}                                                           %
   {$7+3i$}                                                           %
   {$6+4i$}                                                           %
   {$8+2i$}                                                           %
   {They all have the same modulus.}
   \midexamplespacer
                                                                     %E
   %14                                                               %B
   \QQRRS{A circle of radius 1 is centered at the origin.  Starting at
       the point where $\,x=1\,$ and $\,y=0\,$, a distance $\,u\,$ is
       measured along the circle in a counterclockwise direction.  The
       coordinates of the location after moving this distance $u$ are}
   {$x=\sin u,\ \ y=\cos u$}                                          %
   {$x=\cos u,\ \ y=\sin u$}                                          %
   {$x=\tan u,\ \ y=\cot u$}                                          %
   {$x=\cos u,\ \ y=\tan u$}                                          %
   {$x=\sec u,\ \ y=\csc u$}                                          %
   \midexamplespacer
                                                                     %EL
```

---

1. Which of the four complex numbers listed below has the greatest modulus?

A) $8+2i$ B) $6+4i$ C) $5+5i$ D) $7+3i$

E) They all have the same modulus.

---

2. A circle of radius 1 is centered at the origin. Starting at the point where $x=1$ and $y=0$, a distance u is measured along the circle in a counterclockwise direction. The coordinates of the location after moving this distance u are

A) $x=\cos u,\ \ y=\tan u$ B) $x=\sin u,\ \ y=\cos u$

C) $x=\cos u,\ \ y=\sin u$ D) $x=\sec u,\ \ y=\csc u$

E) $x=\tan u,\ \ y=\cot u$

Figure 1: An Example of part of `TEST.SCR` and the resulting output

```
        character   character      character     character   character
         position    position       position      position    position
           10          21             36            49          59
           ↓           ↓              ↓             ↓           ↓
version  2 scramble 5  answerlines 2  rootlines 3  answer B
The graph of the \U{intersection} of the
   equations\ \ $\cases{\hskip-.12in  &  $x-y=-3$  \cr \hskip-.12in
   &  $x+y=1$ \cr}$\hskip.2in is a point that is located
in Quadrant I.
in Quadrant II.
in Quadrant III.
in Quadrant IV.
on a coordinate axis.
```

(The remaining 6 lines of the 15 lines that form this version are blank.)

```
        character   character      character     character   character
         position    position       position      position    position
           10          21             36            49          59
           ↓           ↓              ↓             ↓           ↓
version  6 scramble 5  answerlines 1  rootlines 1  answer A
$\displaystyle\frac{x(x+5)+2(x+6)}{x+4}\ =$
$x+3$
$x+2$
$x+1$
$x+4$
$x+5$
```

(The remaining 8 lines of the 15 lines that form this version are blank.)

---

1. The graph of the <u>intersection</u> of the equations $\begin{cases} x-y=-3 \\ x+y=1 \end{cases}$ is a point that is located

A) in Quadrant IV. B) in Quadrant III. C) in Quadrant I.

D) on a coordinate axis. E) in Quadrant II.

---

2. $\dfrac{x(x+5)+2(x+6)}{x+4} =$

A) $x+1$ B) $x+3$ C) $x+2$ D) $x+4$ E) $x+5$

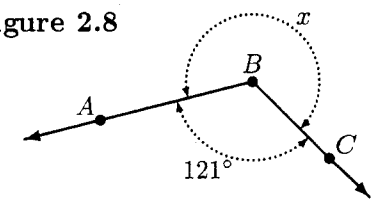
Figure 2: Examples of a version from each of two master questions, and the resulting output

Sometimes knowledge of the measure of one or more angles in a geometric drawing will enable you to determine the measures of other angles. The examples below show some of the techniques that can be used.

---

1. Rays BA and BC determine an obtuse angle having measure 121° (see Figure 2.8 at right). These same rays also form a reflex angle, which has been labeled x in the drawing. From knowledge of the measure of the obtuse angle ABC, determine the measure of the reflex angle ABC.

Figure 2.8



The sum of the measures of the obtuse angle and the reflex angle must be 1 revolution, or 360°. Although x is a name of the reflex angle, we will also let x represent the measure of this angle. Therefore,

$$121° + x = 360° \implies x = 360° - 121° \implies x = 239°.$$

---

2. In Figure 2.9a, angle XYZ is a right angle. Determine the approximate location of point P on the arc, if $\angle XYP$ is to have measure 45°.

Figure 2.9a

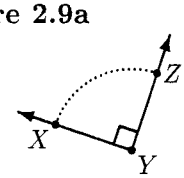
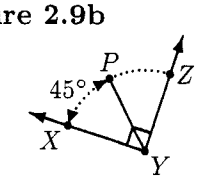


Figure 2.9b



Since 45° is half of 90°, point P must be located so that a rotation of ray YX to ray YP would be half of the rotation of that ray to ray YZ. If P is located halfway along the arc from X to Z (see Figure 2.9b), then the measure of $\angle XYP$ will be half the measure of $\angle XYZ$.

---

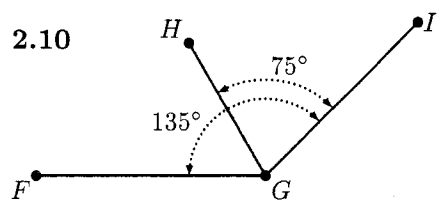
3. In Figure 2.10, these angles are shown:

$\angle FGI = 135°$ $\qquad$ $\angle HGI = 75°$

a) What is the measure of $\angle FGH$?

b) What is the measure of reflex $\angle FGH$?

Figure 2.10



Unless information is given that an angle is a reflex angle, we assume that the angle we are interested in will be the one having the smallest possible measure. In part (a), therefore, $\angle FGH$ is referring to the acute angle, whose measure will be the difference between the measures of the obtuse angle FGI and the acute angle HGI. In part (b), we can determine the measure of the reflex angle by subtracting the acute angle's measure from 360°.

a) $\angle FGH = \angle FGI - \angle HGI = 135° - 75° = 60°$

b) reflex $\angle FGH = 360° -$ acute $\angle FGH = 360° - 60° = 300°$

Figure 5: Example of part of a page from *Essential Geometry* that made use of `ANGLEARC.TRU`

T-EDIT, A Collection of Editing Macros for TEX

Larry F. Bennett
Department of Mathematics
South Dakota State University
Box 2220
Brookings, South Dakota 57007-1297
Phone: 605-688-6218
Bitnet: MA01@SDSUMUS.BITNET

Abstract

T-EDIT is a powerful collection of editing macros designed specifically for TEX. The macros are designed to be used with the KEDIT editor on IBM PC or PC-compatible computers. The user is aided in every step of the preparation of a completed document. Menus or prompting messages are included with many of the macros. Over 1250 TEX and $\mathcal{A}\mathcal{M}\mathcal{S}$-TEX control sequences may be accessed through menus. The control sequences may either be inserted directly into the text or assigned to function keys. T-EDIT can be used to control the TEXing and possible previewing of output. Debugging features are included. Macros generating several lines of complicated TEX source code are available, and TEX macros have been designed to be used with several of the code-generating macros.

Introduction

T-EDIT is a powerful collection of editing macros, TEX execution control macros, and a collection of special TEX macros to be used in conjunction with the other macros. The macro package is designed to be used with the KEDIT editor on IBM PC or PC-compatible computers, but can probably be modified to be used with other editors in other environments. In addition, although it currently makes use of a PCTEX implementation of TEX, the ArborText previewer, and the MicroSpell spelling checker, T-EDIT is in no way bound to these software packages. With a few simple changes, it should work with other software.

The editing macros are currently all written in the macro language KEXX, which is essentially a subset of the computer language REXX.

The T-EDIT macro package was developed to make the entire process of creating a TEX source program, TEXing it, previewing it, and debugging it into a fairly simple task. This has been accomplished in many instances by using menus and prompting messages within the macros which guide the user through each required step. Like TEX macros, existing T-EDIT macros may be modified if desired, and new T-EDIT macros may be created. In addition, menus may be modified and created.

The present version of T-EDIT was designed under the assumption that its primary use would be for the entry of mathematical text into source files. Consequently, many T-EDIT macros have been created for specific mathematical applications. Furthermore, the macros have all been designed under the assumption that the user will employ Plain TEX and/or $\mathcal{A}\mathcal{M}\mathcal{S}$-TEX. However, since T-EDIT may be easily modified and expanded, it would be a simple matter for a user to modify it to suit his or her needs.

An editing macro package for TEX based upon the KEDIT editor was developed in 1989 by Don L. Riley and Brad L. Halverson (See Riley and Halverson). These macros were based mainly upon REXX macros instead of KEXX macros. T-EDIT was developed independently. Consequently, the two collections of macros are completely different and were created with different needs in mind.

In the remainder of this paper, I shall describe what T-EDIT is capable of doing, although I will *not* be able to discuss everything. In most cases, I will not be able to go into too much detail about how it accomplishes what it does. This is simply because of the enormous number of the macros employed by T-EDIT, the length of some of the macros, and the fact that the macros are written in a computer language which most readers may not be familiar with. This paper should be considered as an *introduction* to T-EDIT.

Editing Features

Special keys. T-Edit employs special keys and key combinations to implement KEXX macros. IBM compatible Personal Computer keyboards are ordinarily supplied with ten so-called *Function* keys, which are designated on the keyboard as `F1`, `F2`, ..., `F10`. In addition, keyboards which are referred to as *extended keyboards* have two additional Function keys: `F11` and `F12`. In this paper, it will be assumed that an extended keyboard is being employed. There is also a *Control* key, which is denoted by `Ctrl`, and there is an *Alternate* key which is abbreviated `Alt`. *Key combinations* may be formed using `Ctrl` and `Alt`. To form a key combination using `Ctrl`, the `Ctrl` key is held down and a second key is pressed. If the second key happens to be the Function key `F5`, then `Ctrl-F5` is used to denote this key combination. Similarly, if `Alt` is held down and `P` is pressed, then this gives the key combination `Alt-P`. Any one of the key combinations `Alt-F1`, `Alt-F2`, ..., `Alt-F12` will be referred to as an `Alt-F` key. The purpose of the `Alt-F` keys will be discussed later. The Escape key, which is denoted by `Esc`, is another special key that is employed in various T-EDIT macros.

KEDIT allows a user to assign a KEXX macro to any key or combination of keys, and the macro is stored in computer memory. Such a macro will be referred to as both a *level-0* macro and as the macro *associated with* the key or combination of keys. KEDIT refers to the associated macro using the name of the key or key combination it is associated with. For example, the macro associated with `Ctrl-T` is named `Ctrl-T` by KEDIT. This naming convention makes it possible to initiate the macro without pressing the corresponding key or combination of keys. For example, including the command `'macro Ctrl-T'` in another KEDIT macro will cause the macro associated with the key combination `Ctrl-T` to be executed.

Throughout this paper, a *simple macro* should be thought of as level-0 macro which does *not* call upon any macro stored on the hard disk. A *redirection macro* is a level-0 macro which calls immediately upon a macro stored on the hard disk. Ordinarily, T-EDIT uses the same name for the macro on the hard disk as the name of the redirection macro. For instance, `Ctrl-T` is a redirection macro, and it calls immediately upon the macro `Ctrl-T.kex` on the hard disk. The extension `kex` is simply the default extension employed by KEDIT for macros stored on the hard disk. Whenever a macro which resides on the hard disk is called upon, it is loaded into memory, executed, and then released from memory. These macros may also call upon other macros stored in memory or on the hard disk. Any macro T-EDIT associates with a key or combination of keys is either a simple macro or a redirection macro. However, T-EDIT makes use of many more redirection macros than simple macros, and throughout the remainder of the paper, all macros associated with keys or key combinations should be assumed to be redirection macros unless specified otherwise.

KEDIT only allows macros to be associated with a single key or a key combination consisting of two keys. In order to create the illusion of *extended key combinations*, T-EDIT makes use of selection macros. By definition, a *selection macro* is a macro which is stored on the hard disk whose *only* purpose is to allow the user to select from other macros which are stored on the hard disk by simply pressing a key or combination of keys.

Although several selection macros are employed by T-EDIT, only three are going to be mentioned in this paper. As mentioned earlier, the redirection macro `Ctrl-T`, associated with the key combination `Ctrl-T`, calls upon the macro `Ctrl-T.kex` which is stored on the hard disk. The macro `Ctrl-T.kex` is a selection macro. Once this macro is initiated, a message appears on the screen informing the user of possible choices of keys to press next. One of the options which is available, and which will be discussed in more detail later, is to press `0`. The character associated with the key pressed is read into a KEDIT variable called `readv.1`. Next, a substitution is made in a KEDIT command which is similar to, but slightly more complicated than the KEDIT command

```
'macro \Ctrl-T'readv.1'.kex'.
```

Thus, the macro called upon at this point is actually `Ctrl-T0.kex`. Although there is no message to the effect that it is possible to enter the extended key combination in any other way, the same results are achieved by pressing `Ctrl-0` instead of `0`. This option is allowed since after pressing `Ctrl-T` it is very easy to forget and press `0` next while still holding down the `Ctrl` key. Because of this additional option, the extended key combination `Ctrl-T` followed by either `0` or `Ctrl-0` will be designated as `Ctrl-T0`. Similar notation will be used to denote all other extended `Ctrl-T` key combinations as well as extended key combinations involving `Ctrl-I` and `Ctrl-D`, which are also associated with redirection macros and which call upon selection macros.

Throughout the remainder of this paper, extended combinations will be used without further explanation.

Automatic key assignments. Whenever a TEX source program is edited with T-EDIT, a great number of macros are automatically associated with keys and key combinations. It is the collection of macros associated with various keys and key combinations which does much of the work for T-EDIT. Some of these macros call upon menus which may be used to select other macros, insert TEX source code, or associate macros with `Alt-F` keys, etc.

Text insertion. Many macros associated with keys, key combinations, or extended key combinations eventually cause text to be inserted in a document. Any time T-EDIT inserts text into a document, it automatically puts the KEDIT editor into *insert mode* and often repositions the cursor in the proper position for the user to continue. This means that text, which was above and to the right of the cursor at the time the new text is inserted, will be moved to the right. Furthermore, until the user turns insert mode off, additional text typed in will cause any characters above and to the right of the cursor to be shoved to the right.

TEX control sequence insertion macros. The macro associated with the key combination `Ctrl-I` is a redirection macro. It calls upon the selection macro `Ctrl-I.kex`. Although many options are available after `Ctrl-I` is pressed, only one will be discussed in this paper. In particular, suppose that @ (that is, `Shift-2`) is pressed next. A menu of over 1250 Plain TEX and $\mathcal{A}\mathcal{M}\mathcal{S}$-TEX control sequences and corresponding text insertion macros is initiated. The screen is split into 2 windows. The source program which is being edited appears in the lower window with the cursor under the character it was positioned under when `Ctrl-I` was initially pressed. The user is in what may now be referred to as *menu mode*.

The user sees the first 9 control sequences of the over 1250 which are available displayed in the upper screen window with the first of these control sequences highlighted in red. Also displayed near the top of that window is a message which lists, in a somewhat cryptic form, eighteen options which are available. One of these options, Name, is included to suggest that the user *begin* to type in the name of the control sequence. Suppose that the control word `\begingroup` is needed. After typing `B` and `E`, the user sees

```
\begingroup  \endgroup
```

highlighted in red. In addition, the characters which have been typed in so far, namely `be`, are displayed in the upper portion of the top screen window in order to keep track of what has been typed in. Note that these are lower case letters. In order to have obtained the corresponding upper case letters, the `Shift` key would have had to have been employed. The cursor is still resting beneath the same character as when `Ctrl-I` was pressed. If `Enter` is pressed at this point, then the text

```
\begingroup  \endgroup
```

will be inserted beginning at the current position of the cursor, and the cursor will be repositioned to rest under the second \. Any characters on the current line which were above and to the right of the cursor are moved to the right. The menu is still displayed in the top screen window, but there is a new message that informs the user that he or she is in *program mode*. This means that editing can proceed as usual, even though it is actually being accomplished under the control of the macro `Ctrl-I.kex`. The message also instructs the user that to enter menu mode again, press `Ctrl-Enter`, or press `Alt-Esc` to get rid of the menu and return the screen to full screen edit mode.

Now, let's assume that once a desired entry is highlighted, then instead of pressing `Enter`, the user presses @ (that is, `Shift-2`). Then the highlighted text insertion macro is assigned to some undefined `Alt-F` key. In the bottom window of the screen, a new message appears which tells what `Alt-F` key the macro was assigned to and describes what the macro does. In the case of the preceding example, the displayed message would be

```
Alt-F3 : 'text \begingroup \endgroup'
```

if the text insertion macro was automatically assigned to the undefined `Alt-F` key `Alt-F3`. The user can prohibit the display of such information, then display it again, etc. This time, T-EDIT stays in menu mode. After all, there are probably more text insertion macros which should be assigned to `Alt-F` keys. To escape this mode, `Esc` may be pressed to get back into program mode. In fact, `Esc` may be used at any time the user is in menu mode to return to program mode.

Additional keys which may be used in the menu mode to make a selection are the `Up Arrow` and `Down Arrow` keys, `PgUp` and `PgDn` keys, and the `Ctrl-PgUp` and `Ctrl-PgDn` key combinations. Pressing the `Backspace` key will undo the last key entry, and

pressing the key combination `Alt-Enter` will cause the highlighted macro or text insertion macro to be executed with an automatic cancellation of the menu.

It should be clear that macros which control menus are not selection macros as described earlier. All but 21 of the over 1250 T-EDIT KEXX text macros which can be accessed using the menu just described are generated on the fly. That is, they do not exist at the time they are called upon, but are created using a special code which is present in the menu line containing the desired control sequence, but which is *not* seen by the user. Specifically, a code appears in the first 4 columns of each menu line which instructs T-EDIT how to handle the creation of the text insertion macro, or if the macro is saved on the hard disk instead, then the code `m` is found in column 1 and the name of the macro, which is visible to the user, is found starting in column 69 of the line. Although the other codes available are quite simple, it is far beyond the scope of this paper to describe them all.

Private macros. The key combination `Ctrl-P` will activate a private menu of user defined macros. This menu in used in the same way as described for the menu of TEX insertion macros. However, many of the macros which may be called upon in this menu are much more powerful than any control sequence TEX insertion macros. In addition, the menu may call upon sub-menus, and a couple of extra options may be used in searching for specific macros. Also, in most cases, a fairly lengthy description of what the macro will do is included, and this will be displayed at the bottom of the screen if the user elects to assign the macro to an `Alt-F` key. Furthermore, when called upon, most of these macros lead the user through all of the additional steps necessary to fill in needed data. Some of these macros are powerful TEX code-generating macros which in many cases make use of specially designed TEX macros.

Examples of private macros. Three examples have been included in the Appendix to demonstrate the capabilities of T-EDIT. In each case, a typesetting problem is presented, the steps necessary to generate the required TEX source code using T-EDIT are discussed, and the output obtained from the code is displayed. Please look carefully at these examples so that you can judge for yourself!

At the present time, the private menu refers to a great number of macros which perform remarkable typesetting feats with a minimum amount of effort. In the future, many new private macros will be added to T-EDIT. As a matter of fact, the list will probably continue to grow indefinitely as more and more applications arise. At the same time, many difficult typesetting problems will be reduced to trivial tasks.

Alt-F key management. If the key combination `Alt-E` is pressed, then a number of options are made available. It is now possible to edit or delete `Alt-F` key macro definitions and descriptions, or even enter new ones directly from the keyboard. *Collections* of `Alt-F` key definitions and descriptions may be automatically saved and added to a menu. When the user saves such a collection of `Alt-F` key definitions, he or she is asked to enter a name for the file in which to store the key information as well as a description of the collection. This is then added to a menu so that the entire collection of key definitions and descriptions may be easily reinstated later using that menu. In addition, files containing collections of `Alt-F` key definitions as well as the menu referring to collections of `Alt-F` key files may be edited. The macros associated with `Alt-E` make each of these tasks fairly easy. Because of the capabilities just mentioned, the key assignments which T-EDIT makes at the beginning of each editing session should be adequate for most users.

Letter and mail merge menu. By pressing `Alt-O`, a menu of letter-writing and mail-merge options appears on the screen. The user is aided in writing documents with a minimum amount of effort. Descriptive information concerning a document is recorded in a menu. Later, if the document needs to be located, the desired menu is called upon. Once the information describing the desired file is seen highlighted in red, pressing `Enter` will cause the file to be loaded into the KEDIT editor to be reviewed or revised. Special data-writing macros for mail-merge documents are included. Again, descriptive information concerning specific data sets is inserted in a menu so that the data set may be located easily in the future. Other options allow letter or data menus to be edited. Of course, all documents and data sets are written for use with TEX.

The key combination `Alt-O`, which may seem somewhat out of place, was chosen so that no useful key combination would be wasted. It was included as an additional way of accessing a macro on the hard disk which was actually designed to be initiated from the DOS command line using the Function key `F11`.

Special text insertion keys. Some examples of simple macros associated with keys or combinations of keys are as follows. Keep in mind that T-EDIT is currently used most frequently to enter text which contains mathematical text written for use with Plain TEX or $\mathcal{A}\mathcal{M}\mathcal{S}$-TEX. It is useful to have additional methods of inserting often-used control sequences so that these control sequences can be entered as easily as possible. When `F1` is pressed, `$$` is inserted in the file beginning at the position of the cursor at the time that the key is pressed. The cursor is positioned under the second `$` sign. The simple macro associated with the Function key `F1` is

```
'insert on';'text $$';'cur left'.
```

Definitions of other simple macros are similar. If the user proceeded to type `E=mc^2`, then after typing the `2`, `$E=mc^2$` would be seen with the cursor positioned under the last `$` sign. Remember, KEDIT was put automatically into insert mode when the `F1` key was pressed. Similarly, pressing `Ctrl-[` causes `{}` to be inserted as text with the cursor repositioned below `}`. The symbol `{` appears on the key referenced by the symbol `[`, so the choice of `Ctrl-[` to represent `{}` is a fairly natural one. Pressing `Ctrl-\` causes `\{ \}` to be inserted into the source file with the cursor resting under the blank space to the right of `{`. Here, since the first character generated is the character `\`, the key combination `Ctrl-\` was chosen to generate this character string.

The macro associated with the key combination `Ctrl-D` is a redirection macro which calls upon the selection macro `Ctrl-D.kex`. When `Ctrl-D` is pressed, a message appears on the screen which instructs the user to press `Esc` to get out of the menu, press `1` for `$$`, press `2` for `$$$$`, or press `3` for `$$ $$` spread out over three lines. Only one of the three options will be discussed. Suppose the user presses 3. Then the following will occur. If the cursor is *not* resting on a blank line, then a new line is first added after the line the cursor was positioned on. Three lines of text are created. The first line contains `$$` left justified, the second line is a blank line, and the third line is `$$`. The cursor is repositioned to appear at the first column of the blank line separating the two `$$` groups. This format is used when mathematical expressions are to be entered in display math style.

The macros associated with the key combinations `Ctrl-G` and `Ctrl-O`, which are discussed next, are redirection macros, but the corresponding macros on the hard disk which they call upon are *not* selection macros or menu-type macros. Details cannot be included in paper as short as the present one.

Greek letters. To get Greek letters quickly, the combination of keys `Ctrl-G` may be pressed followed by one, two, or three of the letters in the control word representing that letter. For example, the text `\varepsilon` is automatically inserted after `Ctrl-T` is pressed followed by `VE`, while the text `\gamma` is automatically inserted after pressing `Ctrl-G` followed by `G`. The instant that the user has typed in enough information to distinguish the desired Greek letter from all others, then the text for that Greek letter is automatically inserted. The user never presses the `Enter` key.

Math delimiters. Opening and closing math delimiters can be obtained by pressing `Ctrl-O`. Suppose that the text

```
\left\lceil  \right\rceil
```

needs to be inserted in a document to create delimiters for a mathematical expression. Assume that the key combination `Ctrl-O` is pressed. A message appears at the top of the screen which instructs the user to press the key which represents the first symbol in the opening delimiter. So suppose that `\` is pressed. At this point, the screen will be split into two windows, with a menu displayed in the upper window. The source file appears in the lower window with the cursor positioned as it was before `Ctrl-O` was pressed. Suppose `LE` is typed next. The code `\left \right` appears highlighted in red in the menu. Assume that `Enter` is pressed. The text `\left` is inserted. However, note that `\right` has not been inserted yet. It has been saved to be inserted later. The menu is still in effect, and a message instructs the user to type the first symbol of the code representing the opening delimiter. That symbol would be `\` for the opening delimiter `\lceil`. So suppose that `\` is pressed next. A matching math opening and closing delimiter code is highlighted in red, but it is not the pair which is currently being sought. However, after pressing `C` and `E`, the desired matching opening and closing delimiter pair is seen highlighted in red. If the `Enter` key is pressed next, then the menu disappears, and it is observed that the TEX code

```
\left\lceil  \right\rceil
```

has been inserted in the source file with the cursor positioned under the first space to the left of the control word `\right`. Note how the control word `\right`, which was not present before, has now been

inserted in the proper position. Incidentally, the requirement that \ be entered as the first symbol in the control word representing the opening delimiter is due to the fact that simple math opening and closing delimiter pairs like () may be used also.

TEXing, Previewing, and Debugging

When a source file is ready to be TEXed, the combination of keys `Ctrl-TE` is pressed. A check is made to make sure that a `\bye` or `\end` statement is included at the end of the program. If there is no such statement, then one is added. The TEX source file is saved, and then TEXed. If no error occurs when the source file is TEXed, then the output is automatically previewed. After the preview is completed, there is a return to the editor and the source file. The cursor is positioned wherever it was when `Ctrl-TE` was pressed. If an error occurs while TEXing the file, and if the user enters the TEX option `e`, then the line in the source file where the error occurred becomes the current line in edit mode, and the cursor is positioned at the beginning of this line. Furthermore, the screen is split into two windows, and as much of the *pertinent* `log` file information as possible is shown in the upper window. If it isn't possible to display all of this information, then the user is advised and informed to press `Ctrl-U` if there is need to review more of the error message than is shown. If `Ctrl-U` is pressed, then although the cursor never moves from its current position in the source file, the `log` file may be reviewed using the `Up Arrow`, `Down Arrow`, `PgUp`, `PgDn` keys, etc. The key combination `Ctrl-TT` may be used to TEX a file without previewing the output.

Pressing the key combination `Ctrl-TB` allows a user to TEX a KEDIT so-called "marked block" in a file. This could include as much or as little TEX source code as desired. In fact, a single character could be TEXed this way. What actually happens is that the block is saved as a TEX source file called `\exper.tex` with a `\bye` statement inserted as the last line. This file is then TEXed and previewed. If there is an error, then the offending line in the original file, not the `\exper.tex` file, becomes the current line in the editor with the cursor positioned at the beginning of that line. The screen is split into two windows, with the error message or a portion of it displayed as was described above.

Pressing the key combination `Ctrl-TO` does the very same thing as `Ctrl-TB` with one important exception. Any lines in the file with `%` in column 80 are also included in the `\exper.tex` file. Such lines can be marked immediately by pressing `Ctrl-M` while the cursor is positioned on the line. Later the symbol `%` may be erased by pressing `Ctrl-E` while the cursor is positioned on the line. However, in most cases, only certain lines need to be marked in this way, and they will not have to be altered again later.

Pressing the key combination `Ctrl-TD` will cause all lines in the current source file, from the first blank line at or above the cursor to the bottom of the file, together with any lines marked with a `%` in column 80, to be TEXed, and previewed, etc.

Pressing `Ctrl-C` will cause a forward search from the beginning of the file for matching pairs of `{}`, followed by a backward search from the end of the file for such matching pairs. Pressing the key combination `Ctrl-TC` followed by `F` will cause a forward search from the current cursor position to the end of the file for matching pairs of this type, while pressing `Ctrl-TC` followed by `B` will cause a backward search from the current cursor position to the top of the file for such pairs. In all cases, the search will end at the first `{` or `}` for which there is no match.

If `Ctrl-S` is pressed, then the MicroSpell spelling checker is activated. Up to about 90% of the spelling errors in the document are usually spotted and corrected this way. The key combination `Ctrl-TP` will cause the output to be printed, and `Ctrl-TH` will cause the `dvi` file associated with the source file to be copied to a diskette.

In case a user cannot recall all of the pre-defined key definitions, `Alt-H` brings up a menu of all such key assignments the user may not be familiar with. Once the given key and description are shown highlighted in red, the key can be accessed by simply pressing the `Enter` key! By pressing `Ctrl-H`, a menu comprised of a subset of the key definitions just mentioned, which are TEX related, is initiated.

Future of T-EDIT

T-EDIT is still in its infancy. It will continue to grow and become more sophisticated in the future regardless of whatever else happens. Hopefully, other individuals will become interested in T-EDIT, and perhaps a method for the distribution of T-EDIT software will become available.

Bibliography

Riley, Don L. and Brad L. Halverson, "Creating an Efficient and Workable PC Interface for TEX", *TUGboat*, **10** (4), pages 751–759, 1989.

Appendix

Example 1

Suppose that a user wants to display the long division of the polynomial

$6x^6 - 15x^5 - 34x^4 + 36x^3 - 14x^2 - 7x + 7$

by the polynomial

$2x^2 + 3x - 2.$

The user presses `Ctrl-P` to get into the private macro menu. After `POL` has been typed in, the user sees the entry which says `polynomial division` highlighted in red. If the user presses `Enter`, then the first thing which is done by T-EDIT is to search the TEX source file for an `\input` statement for file `\polydiv.mac`, which contains the TEX macros which will be needed. If no such `\input` file is found, then one will be added automatically to the source file at the beginning of the file or following the last `\input` statement found, if any. While this is accomplished, the cursor appears to stay in precisely the same position that it was in before `Enter` was pressed. Next, the name of the variable to be used is requested. This could be a Greek letter, etc. if desired. Suppose that x is entered. The user is then asked to insert the coefficients of the denominator. Here, that would mean that `2 3 -2` is typed in and then entered. Now the coefficients of the numerator are requested. For the current problem, the user would type `6 -15 -34 36 -14 -7 7` and enter it. T-EDIT causes all of the additional numerical information required to be computed. Line after line of TEX code is automatically generated until all necessary code has been inserted. The code generated by T-EDIT is shown below.

```
\polydiv
\lp 3x^{4}\m 12x^{3}\p 4x^{2}\p 0x\m 3
\hbar{15}
2x^{2}+3x-2\vbar\lp 6x^{6}\m 15x^{5}\m 34x^{4}\p 36x^{3}\m 14x^{2}\m 7x\p 7\crn{3}
\lp 6x^{6}\p 9x^{5}\m 6x^{4}\ubar{4}{5}{5}
\lm 24x^{5}\m 28x^{4}\p 36x^{3}\crn{5}
\lm 24x^{5}\m 36x^{4}\p 24x^{3}\ubar{5}{6}{7}
\lp 8x^{4}\p 12x^{3}\m 14x^{2}\crn{7}
\lp 8x^{4}\p 12x^{3}\m 8x^{2}\ubar{8}{5}{11}
\lm 6x^{2}\m 7x\p 7\crn{11}
\lm 6x^{2}\m 9x\p 6\ubar{11}{6}{13}
\lp 2x\p 1
\endpolydiv
```

The output which will be produced by the code is shown below.

$$
\begin{array}{r|l}
 & 3x^4 - 12x^3 + 4x^2 + 0x - 3 \\ \cline{2-2}
2x^2 + 3x - 2 & 6x^6 - 15x^5 - 34x^4 + 36x^3 - 14x^2 - 7x + 7 \\
 & \underline{6x^6 + 9x^5 - 6x^4} \\
 & \quad - 24x^5 - 28x^4 + 36x^3 \\
 & \quad \underline{- 24x^5 - 36x^4 + 24x^3} \\
 & \qquad\qquad 8x^4 + 12x^3 - 14x^2 \\
 & \qquad\qquad \underline{8x^4 + 12x^3 - 8x^2} \\
 & \qquad\qquad\qquad - 6x^2 - 7x + 7 \\
 & \qquad\qquad\qquad \underline{- 6x^2 - 9x + 6} \\
 & \qquad\qquad\qquad\qquad 2x + 1
\end{array}
$$

The user can position the output on the page however desired. In the case of the example above, the display has simply been indented by the default indentation amount.

Example 2

A user desires to create a short table of Laplace transforms of functions. Assuming that the appropriate T-EDIT macro has not already been assigned to an `Alt-F` key, the first step in creating the table would be to press `Ctrl-P` to access the private macro menu. After `TA` has been typed, the desired entry is highlighted in red. It is listed in the form `table, math` together with some additional information which identifies the table style. Beneath this entry, other styles of math tables are listed.

To initiate the macro, the user presses `Enter` (or `Alt-Enter` if the menu is to be exited after the selection of the macro). A check is made to see if the user has included an `\input` file for the required TEX macro `\mathtabl.mac`. If no such statement is found, then one is added after the last `\input` statement in the file, or at the beginning of the file if there is no `\input` statement. A beep sounds and a message appears which requests the user to enter the title on one line, separating lines of the title with `\cr`, or to press `Enter` to quit. Suppose that `Laplace Transforms\cr of Functions` is typed in and entered. A new line is added if the line upon which the cursor is resting is not a blank line. The text below is inserted, a beep sounds, and a message appears requesting the user to either enter the amount to spread the table or to press `Enter` to accept the default of 100pt.

```
\mathtable{Laplace Transforms\cr of Functions}
```

Assume that `Enter` is pressed to accept the default. The symbols `{}` are added to the text shown above. There are three more input parameter values which the user may enter or accept default values for. The first of these parameter values is the amount of indentation from the left margin, and has a default value of 1truein associated with it. The other two parameters represent strut heights and depths which will be used in the construction of the main part of the table. Suppose that the default value is accepted for each of these parameters. After the last time `Enter` is pressed to accept a default value, a new line is automatically added, the symbol `|` is inserted, and the cursor is positioned two spaces to the right of it. So far,

```
\mathtable{Laplace Transforms\cr of Functions}{}{}{}{}
  |
```

has been generated. Now, the user is asked to enter a heading for the first column and press the Tab key when the entry is completed. Suppose that `f(t)` is typed, and then the Tab key is pressed. The following text is displayed with the cursor positioned two spaces to the right of the last `|` shown.

```
\mathtable{Laplace Transforms\cr of Functions}{}{}{}{}
  | f(t)                              |
```

Next, a message requesting the entry of the column heading for the second column appears. The message also includes instructions to press the Tab key when the entry is completed. Assume that `{\cal L}(f)` is typed in and the Tab key is pressed. The text which has been inserted after the Tab key was pressed is shown below.

```
\mathtable{Laplace Transforms\cr of Functions}{}{}{}{}
  | f(t)                              | {\cal L}(f)                        |
  |
```

The symbol `|` has been automatically inserted twice more, and the cursor is positioned on the last line shown, two spaces to the right of it. The user is requested to insert the entry for the first column and press the Tab key when that entry is completed, or to press `@` to finish the creation of the table. Suppose that 1 is typed in and the Tab key is pressed. So far, the following text has been added to the TEX source file .

```
\mathtable{Laplace Transforms\cr of Functions}{}{}{}{}
  | f(t)                              | {\cal L}(f)                        |
  | 1                                 |
```

A space separates the last symbol `|` on the right from the current position of the cursor.

The text `1\over s` is typed in next, and the Tab key is pressed. The text below is present after the Tab key is pressed, with the cursor positioned one space to the right of the symbol | on the last line.

```
\mathtable{Laplace Transforms\cr of Functions}{}{}{}{}
  | f(t)                          | {\cal L}(f)                    |
  | 1                             | 1\over s                       |
  |
```

Now that the idea is clear, let's look at the keystrokes needed to complete the table. `Tab` will be used to represent the `Tab` key.

```
e^{at} Tab 1\over{s-a} Tab \sin at Tab a\over{s^2+a^2} Tab \cos at
Tab s\over{s^2+a^2} Tab \sinh at Tab a\over{s^2-a^2} Tab \cosh at Tab
s\over{s^2-a^2} Tab @
```

All of the necessary code to generate the Table has been entered and appears below.

```
\mathtable{Laplace Transforms\cr of Functions}{}{}{}{}
  | f(t)                          | {\cal L}(f)                    |
  | 1                             | 1\over s                       |
  | e^{at}                        | 1\over {s-a}                   |
  | \sin at                       | a\over {s^2+a^2}               |
  | \cos at                       | s\over {s^2+a^2}               |
  | \sinh at                      | a\over {s^2-a^2}               |
  | \cosh at                      | s\over {s^2-a^2}               |
\endmathtable
```

Note that the required control word `\endmathtable` was automatically inserted on the last line. In addition, one extra line has been added following the last line shown, and the cursor is positioned under the first column of that line. The table produced by the code is illustrated below.

Laplace Transforms of Functions

$f(t)$	$\mathcal{L}(f)$
1	$\frac{1}{s}$
e^{at}	$\frac{1}{s-a}$
$\sin at$	$\frac{a}{s^2+a^2}$
$\cos at$	$\frac{s}{s^2+a^2}$
$\sinh at$	$\frac{a}{s^2-a^2}$
$\cosh at$	$\frac{s}{s^2-a^2}$

Example 3

Consider the problem of displaying a linear system of equations. In this example, it will be assumed that the T-EDIT macro has already been assigned to an `Alt-F` key, say `Alt-F5`, using the private menu. Then towards the bottom of the screen, the message

```
*** Use Ctrl-V to toggle info. display. Use Alt-5 to clear keys and screen ***
```

appears together with information for `Alt-F` keys which have been defined. For `Alt-5`, the user would see the following line displayed.

```
Alt-F5 :  systems of equations
```

Assume the key combination `Alt-F5` is pressed. As in the case of the previous examples, the `\input` statement for the macro `\sysequa` is inserted in the source code if it does not already appear. A beep is heard, and a message appears towards the top of the current screen window which instructs the user to enter each variable name to be used *or* enter the name of the subscripted variable name which will be employed followed by the number of unknowns. Pressing `Enter` is mentioned as a method of terminating execution. For this example, suppose that `\alpha 4` is typed in and entered. If the cursor is resting on a line which is not completely blank, then a new line is added. The code

```
\sysequa
```

appears left justified as shown on that line. In addition, a beep is heard and a message appears towards the top of the current screen window which instructs the user to enter four coefficients, followed by the entry to appear on the right-hand side of the equals sign for this row equation. Entering too little information or too much information for this row equation will cause several beeps to sound, together with a message instructing the user to enter values for this row equation again. If `1 35 -3 2 6` is entered then the following code will be seen.

```
\sysequa
  \lp \alpha_{1} \p 35\alpha_{2} \m 3\alpha_{3} \p 2\alpha_{4} \eq{6}
```

Of course, a beep is heard and the same message appears as before. Continuing, let's assume that `15 -4 21 9 5` is entered, followed by `-2 0 2 4 8`, and then `1 3 0 -1 10`. Finally, `f` is entered to finish. The code which has been generated by T-Edit is shown below.

```
\sysequa
  \lp \alpha_{1} \p 35\alpha_{2} \m 3\alpha_{3} \p 2\alpha_{4} \eq{6}
  \lp 15\alpha_{1} \m 4\alpha_{2} \p 21\alpha_{3} \p 9\alpha_{4} \eq{5}
  \lm 2\alpha_{1} \zero \p 2\alpha_{3} \p 4\alpha_{4} \eq{8}
  \lp \alpha_{1} \p 3\alpha_{2} \zero \m \alpha_{4} \eq{10}
\endsysequa
```

Note that `\endsysequa` has been automatically added to the code. Furthermore, an additional line has been added following this line, and the cursor appears at the first column of the new line.

The system of equations generated may be displayed wherever desired on the page. If `$$` had been typed in before the macro was initiated, and if `$$` had been entered on the line following the command word `\endsysequa`, then the output obtained would be the same as that displayed below.

$$\begin{aligned}
\alpha_1 + 35\alpha_2 - \ 3\alpha_3 + 2\alpha_4 &= \ 6 \\
15\alpha_1 - \ 4\alpha_2 + 21\alpha_3 + 9\alpha_4 &= \ 5 \\
-\ 2\alpha_1 \qquad\quad + \ 2\alpha_3 + 4\alpha_4 &= \ 8 \\
\alpha_1 + \ 3\alpha_2 \qquad\quad - \ \alpha_4 &= 10
\end{aligned}$$

Automatic Tables Using SGML, C, and TEX

Robert McGaffey
Oak Ridge National Laboratory
Building 2506 MS 6302
P. O. Box 2008
Oak Ridge, TN 37831-6302 U.S.A.
Phone: 615-574-0618; FAX: 615-574-6983
Internet: rwm@ornl.gov

Abstract

This paper presents yet another method for doing tables with TEX. This process is different in that the source of the tables is not TEX but SGML, and there are no formatting instructions in the input. We present a method whose goal is to produce the highest quality output under the constraints given. We also present a set of problems that result from this method and suggest a solution.

Automatic Tables

Definition. Perhaps the best way to define an automatic table is to describe how it gets created. Suppose you are a TEX expert and I tell you that I want you to typeset a table with a specific number of columns. I also give you the default type of entry for each column. I am not telling you how many rows there are, nor how wide any of the entries might be. Now, it's up to you to design the table without the entries. I will then add the entries myself and expect some good result. Finally, after I add the entries and print the result, I may not modify such things as `\tabskip` glue to make the table be more pleasing to the eye. So, the table is created from its content to the paper without human intervention.

Corollaries. Some obvious conclusions about such tables: since we do not know the length of the headings or the text entries, the decision to wrap headings and/or text must be made automatically. Since we do not know if the headings to a particular column are wider than the other column entries we cannot adopt a simple `\hidewidth` approach for fear that our column headings will overlap each other. Furthermore, we do not know beforehand how long a table will be. Some of ours are 75 pages long which prohibits the direct use of `\halign` due to memory constraints.

Why automation and why TEX. Our problem is that our articles are ultimately to come from a database where a researcher may have selected various parts of many articles for closer scrutiny. In spite of the fact that many want to remove paper from our desks it has not happened yet and will not for some time. Thus our researcher wants several copies of the parts of articles he/she has selected on paper for later study. We do not want anyone at this point to have to gather all of the parts together and typeset them before our hero can have them. We want him/her to be able to say, `PRINT` this and turn around to the laser printer (in a few short minutes) and retrieve the output. Thus our hero need not know any typesetting language at all to get the results. And that's why TEX is being used. It is the most programmable of all typesetters and thus the choice most likely to generate a 'pleasing to the eye' paper for our hero to study.

Goals

We want the following:

1. The inputter should be faced with an easy-to-edit ASCII file.
2. Decimal alignment capability without "extra" columns.
3. No heading to migrate into the heading on either side of it as could happen if `\hidewidth` is used.
4. Sizing of headings and text entries to be determined on the fly depending on the discovered width of both the heading and the column entries under the heading.
5. Both column spanning and row spanning.
6. Tables of many pages to be handled without losing either the inherent benefits of `\halign`'s measuring or exceeding TEX's memory capacity.

7. Headings must be automatically migrated from page to page of a many-page table.

The Algorithm

What I have done. We start with an SGML instance file. For those of you not in the know, this is an ASCII file in which all of the elements of information in a document are labeled with SGML tags. In our case, we use the tag `<CELL>` to indicate a table entry. For example, a numerical field may be indicated by `<CELL>493.7</CELL>`, an equation by `<CELL TYPE="eqn">xy/a</CELL>`, and text by `<CELL TYPE="text">Robert</CELL>`. In the event that a particular cell spans, say three columns and two rows, it must be tagged differently, say, `<CELL TYPE="text" C="3" R="2">Robert</CELL>`. Note that `TYPE`, `C`, and `R` are called attributes of the element `CELL`. A row consists of the start tag `<ROW>`, any number of cells, and the end tag `</ROW>`.

Headings are given special treatment in our system. A main heading consists of `<CH1>`, the heading, and `</CH1>`. Subheadings are included *inside* of the main heading they modify. Let's say we have a table with the main headings: Main 1 and Main 2. And that both of these headings have two subheadings with obvious names. Then the markup of these headings could be `<CH1>Main 1<CH2>Sub 1a</CH2> <CH2>Sub 1b</CH2></CH1><CH1>Main 2<CH2>Sub 2a</CH2><CH2>Sub 2b</CH2></CH1>`. Note that the headings do not come out in the order needed by TEX.

And I could go on to describe the rest of the table elements but I hope the above suffices to show how the information is delineated in the tables we use. If not, the Appendix contains a short sample table and the SGML markup we use for that table.

Each `<CELL>` comes equipped with another attribute not yet shown. The `COORD` attribute gives the row and column numbers spanned by the particular entry. Thus if our large `CELL` above was entered in the fourth row and the third column of a table it could be fully described by `<CELL TYPE="text" C="3" R="2" COORD="4-5,3-5">Robert</CELL>`, as it spans rows four and five, and columns three through five. We do not want inputters to have to deal with the determination of the `COORD` attribute so we have a parser/translator program and a `C` program which together generate the `COORD` attributes. The translator expands the table so that all implied (defaulted) attributes are available to the `C` program which then calculates and outputs the `COORD` attributes.

Now, we have the SGML instance file we wanted in the first place. The integrity of the information has been preserved and the coordinates of table cells are now available for insertion into a database so that our hero may access, say, the third and fifth columns of a table while ignoring the other columns.

Now we use our parser/translator to convert from SGML to pseudoTEX. We are now left with many holes in our TEX file. Since we input subheadings as part of our heading elements, our table headings are interleaved; that is, some subheadings may appear in the document before all of the major headings. Our table entries may have white space before and/or after them. And, we have no preamble.

Recalling our goals, we want to take advantage of the `\halign` ability to premeasure columns and yet also handle 75 page tables. Well, the only way to measure the columns is to let TEX do it. Yet we cannot turn TEX loose blindly or we may someday exceed its memory capabilities. Also, we do not ever want to encounter the extra white space in the last column that Knuth refers to on page 247 of *The TEXbook*. What if we let TEX measure each table entry and report the result to another program which can then decide the best width, height, and depth for each column and row in the table? Then, that program could generate a suitable TEX file which would then typeset the table using an algorithm designed to create a result which is pleasing to the eye.

That's the plan.

So, we execute a `C` program to unravel our headings and to remove unwanted white space. Also, this is where we add our extra columns to the aligned decimal columns so that decimals line up. Note this does not break goal number two, as the inputter never sees this TEX file. The result is a TEX file which will run with the proper macros; but, it will not produce a table. There is still no preamble. What it does produce is a file giving the height, width, and depth of every single cell entry in the table. For the headings and text entries, it assumes that the heading/entry will be typeset on one line.

What I hope to do. Here is where I will use a `C` program to mimic TEX's `\halign` process. The natural width of every single column will be determined in two ways: first, by ignoring all headings, and secondly, by assuming headings are not stacked. When headings are ignored, text columns will also be ignored. At the same time,

the natural height and depth of each row will be determined, also, in two ways. During this phase, every spanning entry will be stored away in a list for future processing.

Next, the width of the text columns and the headings will be determined by trying to mathematically choose widths to make the table pleasing to look at. For example, we don't want a column with a two inch heading sitting on a column of numbers whose natural width is one-half inch. I intend to use a set of parameters which can be tweaked with experience until a good job is done. We shall see.

Each entry consists logically of a cell we can create in TEX with the box `\hbox{\tskip\cskip \vbox{\rskip` cell entry `\rskip}\cskip\tskip}`. The `\tskip`s take the place of `\tabskip`s. `\cskip`s are used to surround the columns of a cell when the header is larger than the rest of the column. `\rskip`s are used to handle vertically spanning problems. And `\vtop` or `$\vcenter$` will take the place of `\vbox` when appropriate.

Now, we have to process our spanning entries. If the entry already fits inside the rows and/or columns it spans, then we remove it from the list. If not, we calculate a row factor which is the amount we must expand the spanned rows to make our spanner fit; or, we calculate a column factor; or, both. Then, since spanning entries can overlap, we are going to select the smallest factor we find and increase each of the spanning rows or columns. This is done by increasing the row's `\rskip`s or the column's `\cskip`s. Now, we reprocess the spanning entries, since the factors may be changed by the previous expansion. We then iterate this process until all of the spanning cells fit. Now we can modify our aformentioned pseudoTEX file by inserting `\settabs` and `\+`s in the right places. We only have to make sure that we have correctly specified each box and that we have left gaps to handle vertical spanning.

Of course, we are talking theory here, not reality yet.

The Problems

The biggest problems here involve the fact that a `C` program that doesn't know TEX is stuck in the middle of the process. For example, we allow footnotes in our tables. If the inputter inserts a period in the footnote, then the `C` program may pick that period to be the decimal upon which it must align the column. I know this from experience. I have not solved this problem in theory or practice. For now, I ignore them. I suspect that if I cause `<FTNOTE>` to be translated to something like `\footnoteiii` and also output the footnote to another file so that the `C` program could never see it, I could make it work.

The Proposal

The real cure, however, is for someone with more smarts and time than I have to tackle this problem without the use of the `C` program. After all, TEX always knows more about the text and math and the curly braces, etc., than I could ever teach a `C` program.

Appendix

Sample Table

XYZABC

XYZ			ABC		
X	Y	Z	A	B	C
372.466	493.7	45	124	489	280
372.40	493.7	45	124	489	280
372.	493.7	45	124	489	280
XY/A		832	abc	774	INT
		qrr	aaa	799	

SGML Representation of Table

```
<TABLE NUMBER="1" ID="xyzabc">

<TITLE>XYZABC</TITLE>

<CH1>XYZ<CH2>X</CH2><CH2>Y</CH2><CH2>Z</CH2></CH1>

<CH1>ABC<CH2>A</CH2><CH2>B</CH2><CH2>C</CH2></CH1>

<TBODY>

<ROW1><CELL>372.466</CELL><CELL>493.7</CELL><CELL>45</CELL><CELL>124
    </CELL><CELL>489</CELL><CELL>280</CELL></ROW1>

<ROW1><CELL>372.40</CELL><CELL>493.7</CELL><CELL>45</cell><cell>124
    </cell><cell>489</cell><cell>280</cell></ROW1>

<ROW1><CELL>372.</CELL><CELL>493.7</CELL><CELL>45</CELL><CELL>124
    </CELL><CELL>489</CELL><CELL>280</CELL></ROW1>

<ROW1><cell type="eqn" c="2" r="2">XY/A</CELL><CELL>832
    </CELL><CELL TYPE="TEXT">abc</CELL><CELL>774
    </CELL><CELL TYPE="TEXT">INT</CELL></ROW1>

<ROW1><CELL TYPE="TEXT">qrr</CELL><CELL TYPE="TEXT">aaa
    </CELL><CELL>799</CELL><CELL></CELL></ROW1>

</TBODY>
</TABLE>
```

Dotex — Integrating TEX into the X Window System

Anthony J. Starks
Merck Research Laboratories
Computer Resources, RY86-200
126 Lincoln Avenue
Rahway, NJ 07065 U.S.A.
Phone: (908) 594-7288
FAX: (908) 594-1455
Internet: ajs@msdrl.com

Abstract

Dotex is a system to integrate TEX tools and automate the format, edit, preview, and print cycle in the X Window System. Dotex uses a single X Window client, xtmenu, to provide a simple push-button interface to the TEX formatter, text editor, and dvi previewer. Other functions such as spell-checking are easily added. Dotex's function is to integrate, not change existing tools, providing a highly interactive "point and shoot" interface to traditional batch-oriented programs. Dotex's chief advantage is allowing the user to rapidly visualize changes in the manuscript, thus facilitating such things as prototyping different typographic effects.

Introduction

TEX is an interactive, terminal based program, but today's computing environment is increasingly window- and mouse-based. Dotex provides a "point and click" wrapper around TEX and other tools in the X Window System (Gettys et al, 1990).

Dotex was built out of the frustration of constantly running an editor and the TEX formatter during the document preparation cycle. The first step was to use the built-in history and line editing functions of the shell command interpreter, but even this was unsatisfactory. What was needed was a tool that would centralize all the actions needed to prepare a document, without having to change the tools themselves. Out of this need, dotex was born.

The Tools

The tools needed to implement dotex are the front-end client, xtmenu, the dvi previewer xdvi and a text editor. The standard X terminal client, xterm is used to start dotex and serves as a logging device.

xtmenu. The X client, xtmenu is the heart of dotex. Xtmenu's function is to bind actions to buttons. When a button (or keyboard equivalent) is pressed, a user-defined action takes place. The action can be some window-management task, or more importantly, the execution of an arbitrary program.

xdvi. xdvi is a typical dvi previewer that has several attractive features:

- It uses the same font bitmaps as the printer, so that a special set of font bitmaps is not needed just for the previewer.
- xdvi has a rich, unobtrusive interaction model for moving around within the dvi file, including jumping to a particular page and zooming to different magnifications. A pop-up magnifier for examining fine typographic details is also provided.
- Also, recent patches provide much improved display quality through anti-aliased font display on color or grayscale X servers.

But the critical feature needed for the success of dotex is xdvi's ability to refresh its display when the dvi file changes.

The editor. Any standard text editor may be used with dotex. Popular editors usually found in a X Window environment such as Emacs and *vi* will work fine. The editor must be able to write its buffer without quitting to be effective with dotex. Window based editors such as xedit are particularly effective because they don't need to be run within an xterm, and already support mouse-based interaction. The author's personal favorite is mx, a programmable mouse-based editor based on

the Tcl language (Ousterhout, 1990). Mx provides mouse-based selection, multiple windows, and other niceties such as support for regular expressions. Another welcome feature for TEX use is the visual indication of the nesting level of delimiter pairs like { } and [ ].

Other tools. Any program that can aid in document preparation can potentially be used with dotex. For example, spell checking can be added by running an interactive speller such as ispell. In the same way, TEX manuscripts may be checked for grammatical correctness.

Configuration

The dotex shell script. A simple xtmenu script looks like this:

```
 1. #
 2. #
 3. #  dotex -- automate TeX and tools
 4. #
 5. #  Usage: dotex file
 6. #
 7. file=$1
 8. xtmenu -noquit -stdin <<!
 9. "$file.tex" !label
10. TeX          %tex $file.tex
11. Edit         %mx $file.tex&
12. Preview      %xdvi $file.dvi&
13. Print        %dvidsk $file|lp -o nobanner -r
14. Done         !exit
15. !
```

Figure 1: The dotex script

It is a UNIX shell script that defines the buttons in the main xtmenu window. Line 7 saves the name of the file that is to be used later in the script. Line 8 is the execution of the xtmenu program itself. The option -noquit tells xtmenu not to quit after an action executed, and the -stdin option means take the input from the standard input file; in this case the shell *here document* (Kernighan and Pike, page 94) contained in lines 9 – 14. If desired, the buttons may be arranged horizontally by adding the -horizontal option.

The lines in the *here document* are the heart of dotex. Each line is a label/action pair. The first string in the pair is a label that manifests itself as either a button or label in the xtmenu menu. The second string is either a keyword defining a label, a system command, or special action. The label/action pair in line 9 defines a label at the top of the menu. This label is for identification only and no action takes place when it is clicked with the mouse. Lines whose action-string begin with % are sent to the command interpreter for execution when their labels are "pressed" by placing the mouse cursor over them and pressing the first mouse button.

Lines 10 – 13 run TEX, the editor, the previewer, and dvi print tool respectively on the target file. The action in line 14 terminates a dotex session, and line 15 terminates the *here document*.

The order of the buttons is arbitrary, but it is best to arrange the most used buttons at the top, and to group the functions logically.

Note that the definitions of the buttons are not necessarily static. With appropriate X resource settings, a given key sequence may be defined which invokes a dialog box in which a new action and/or label can be defined. This mechanism can be used to alter command names or parameters.

A dotex Session

To begin a dotex session, type the command:

```
dotex file
```

in a xterm window running a shell. To make effective use of screen space, the shell window should be about 10 lines long.

This creates the dotex menu, with its buttons corresponding to the script described above:

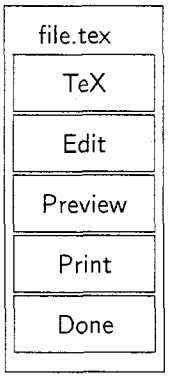


Figure 2: The dotex menu

Generally a dotex session will have these windows on the screen concurrently:

- the dotex menu,
- the editor window,
- the previewer window, and
- the shell, or log window.

The editor and preview windows correspond with the Edit and Preview buttons. The shell window is the xterm used to start dotex. The standard TEX dialog and other system messages are

displayed here. If the operating system supports it, `dotex` can be placed in the background*, and then other commands may be issued in the shell window.

Editing. Pressing the `Edit` button creates an editor window acting on the filename specified in the `dotex` command line with `.tex` appended to it. This is usually the first action. When the file is ready to be formatted by TEX, the edit buffer is written.

Formatting. The next action is running TEX on the file just written by simply pressing the `TeX` button on the `dotex` menu. The normal TEX dialog appears in the shell window.

Note that it is best to position the editor window close to the `dotex` menu so that the operation of writing the file and formatting requires minimal mouse movement.

To achieve tighter integration of the edit-TEX cycle, the act of writing the editor buffer could trigger automatic formatting by making the action of the `TeX` button execute this shell script:

```
touch $1.tex
while :
do
    if newer $1.tex $1.dvi
    then
        tex $1.tex
    fi
    sleep 5
done
```

Figure 3: Triggered TEX script

which automatically runs TEX when the source file is newer than the corresponding `dvi` file. The responsiveness of the formatting is controlled by changing the number of seconds in the `sleep` command. As an extra bonus, the file would also be formatted after any other action that accessed the TEX source such as spell checking.

Previewing. If the TEX run produced no errors, you can preview the file by pressing the `Preview` button. This will bring up a window representing the output of your TEX run. You can move through the `dvi` file, and perhaps magnify sections of the output.

Printing. When you are ready to print, simply press the `Print` button in the `dotex` menu. This will run the `dvi` processor defined in the `xtmenu` script on the file just formatted and previewed. Any messages generated by the `dvi` driver appear in the shell window.

Once the windows are arranged, you are free to move between them, interacting with each as appropriate.

Arranging the windows. The figure in the appendix shows a typical arrangement of the windows. The previewer window is the largest with the other windows atop it. The shell window is kept small and unobtrusive near the bottom of the screen. The `dotex` menu is also near the `Control` menu of the editor, so that writing and TEXing require minimal mouse movement. Note that to preserve screen space it is sometimes useful to iconify both the shell and previewer window. In this arrangement, the focus is on the edit window and the `dotex` menu. The usual interaction is to make changes in the edit window, press the `TeX` button, and then view the file by pressing the mouse on the iconfied previewer window. Alternatively, the file can be previewed by bringing the large previewer window to the front of window stack. When the view is no longer needed, the previewer window can be pushed to the bottom of the stack.

The use of the X Window *window manager*, (Gettys, pages S2/49 – S2/52) is important to the use of `Dotex`, since the window manager's job of window sizing, movement and arrangement effects the productive use of the system. At a minimum the window manager is used to *raise* (place a window on the top of the stack) and *lower* (push a window to the bottom of the stack) the previewer window. All of these actions force `xdvi` to re-read the `dvi` file and present a fresh display.

Any standard X Window window manager will work with `dotex`, but since the predominate actions are raising, lowering and iconifying windows, a minimalist setup with the `twm` window manager (Querica and O'Reilly, chapters 3 and 10), works well. This setup is based the work of Pike (page 284, 1988) and presents a simple pop-up menu of window management functions that look like this:

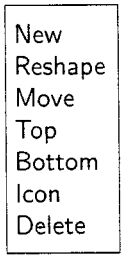


Figure 4: Window Management Popup

* Some implementations do not correctly handle errors during a backgrounded run.

These commands in the twm startup file define the setup:

```
NoTitle
Button3 = :all: f.menu "winmgr"
menu "winmgr"
{
    "New"        !"xterm  -ls &"
    "Reshape"    f.resize
    "Move"       f.move
    "Top"        f.raise
    "Bottom"     f.lower
    "Icon"       f.iconify
    "Delete"     f.delete
}
```

Figure 5: twm Window Management Definitions

Benefits

Prototyping typographic effects. The arrangement just presented has many benefits over the traditional TEX interaction model. For example, it facilitates the rapid prototyping of different typographic effects before final printing. As a simple example, to see the page-breaking effects of altering a parameter such as page width, the steps are:

- go to the editor window, and the add the change, for example, \hsize=3in,
- write the edit buffer,
- go to the dotex menu, press the TeX button, and
- then expose the previewer window.

If the editor supports the undo function, the change can be easily backed off if the desired effect did not meet expectations.

Network usage. Dotex runs in the X Window system so it is inherently network-based. This means that the appropriate hardware can be used for a particular document preparation task. For example, it is possible to run TEX on fast machine and at the same time use a different machine that has fast graphics for previewing.

Reduced loading. Once all of the tools are loaded and on-screen, they remain available to be used when needed, avoiding repeated startup and shutdown.

Rapid correction of errors. Because the error context from the log window and the editor are both visible, it makes error correction easier, not to mention having decreased overhead over the traditional method of having TEX spawn a new instance of the editor. The normal mode of operation becomes the more productive edit-TEX-preview instead of edit-TEX-print.

Enhancements

The window shell wish (Ousterhout, 1991) could replace xtmenu as the front-end to TEX and associated tools. Wish provides a richer widget set as well as the ability to define behavior of applications with the Tcl language. For example, instead of relying on shell scripts to provide the edit-format integration, a Tcl-aware editor such as mx could communicate directly via the send (Ousterhout, 1990) command with a Tcl-aware previewer.

Related Work

Pike (1984) describes an edit-format-preview tool built on the Blit window system, troff and UNIX pipes. This scheme has the advantage of immediate feedback — as soon as the file is written, it is immediately formatted and presented. Pike's approach is to use the operating and window system facilities to build the tool without changing the basic tools.

VORTEX (Chen 1988, pages 133 – 152) takes the approach of an incremental formatter/editor system using TEX and Emacs. The system provides two distinct but integrated views into the document, allowing direct manipulation of both.

Availability

All of the tools needed to implement dotex are publically available on the Internet.

Both xdvi and xtmenu have been posted to the USENET newsgroup comp.sources.x, and are archived on ftp.uu.net in

- packages/X/contrib/xdvi.tar.Z, and
- packages/X/contrib/xtmenu_1.1.tar.Z.

xdvi is also available on export.lcs.mit.edu in the file contrib/xdvi.tar.Z.

The mouse based editor, mx is available on sprite.berkeley.edu in the file tcl/mx.tar.Z

Bibliography

Chen, Pehong, "A Multiple-Representation Paradigm for Document Development", *Technical Report UCB/CSD 88/436*, University of California, Berkeley, July 1988.

Gettys, Jim, Philip L. Karlton, and Scott McGregor, "The X Window System, Version 11", *Software Practice and Experience*, vol. 20, no. S2, pages S2/35 – S2/67, October, 1990.

Kernighan, Brian W. and Rob Pike, *The UNIX Programming Environment*, Englewood Cliffs, NJ: Prentice Hall, 1984.

Ousterhout, John K., "Tcl: An Embeddable Command Language", *Proceedings of the Winter 1990 USENIX Conference*, Washington, D.C., January 22-26, 1990.

Ousterhout, John K., "An X11 Toolkit Based on the Tcl Language", pages 105-115. *Proceedings of the Winter 1991 USENIX Conference*, Anaheim, CA,

Pike, Rob, "The Blit: A Multiplexed Graphics Terminal", *Bell Labs Tech. J.*, vol. 63, no. 8, Part 2, pages 1607 – 1631, 1984.

Pike, Rob, "Window Systems Should Be Transparent", *USENIX Computing Systems*, vol. 1, no. 3, pages 279-296, Summer, 1988.

Quercia, Valerie and Tim O'Reilly, *X Window System User's Guide, for X11 R3 and R4 of the X Window System*, Sebastapol, CA: O'Reilly and Associates, Third Edition, 1990.

Taylor, William, "xtmenu — An X Windows Menu Program", Version 1.0, *comp.sources.x*, vol. 13, xtmenu, June 17, 1991.

Vojta, Paul, et al, "xdvi — DVI Previewer for the X Window System", *comp.sources.x*, vol. 17, xdvi, March, 1992.

Appendix I — A Sample Session

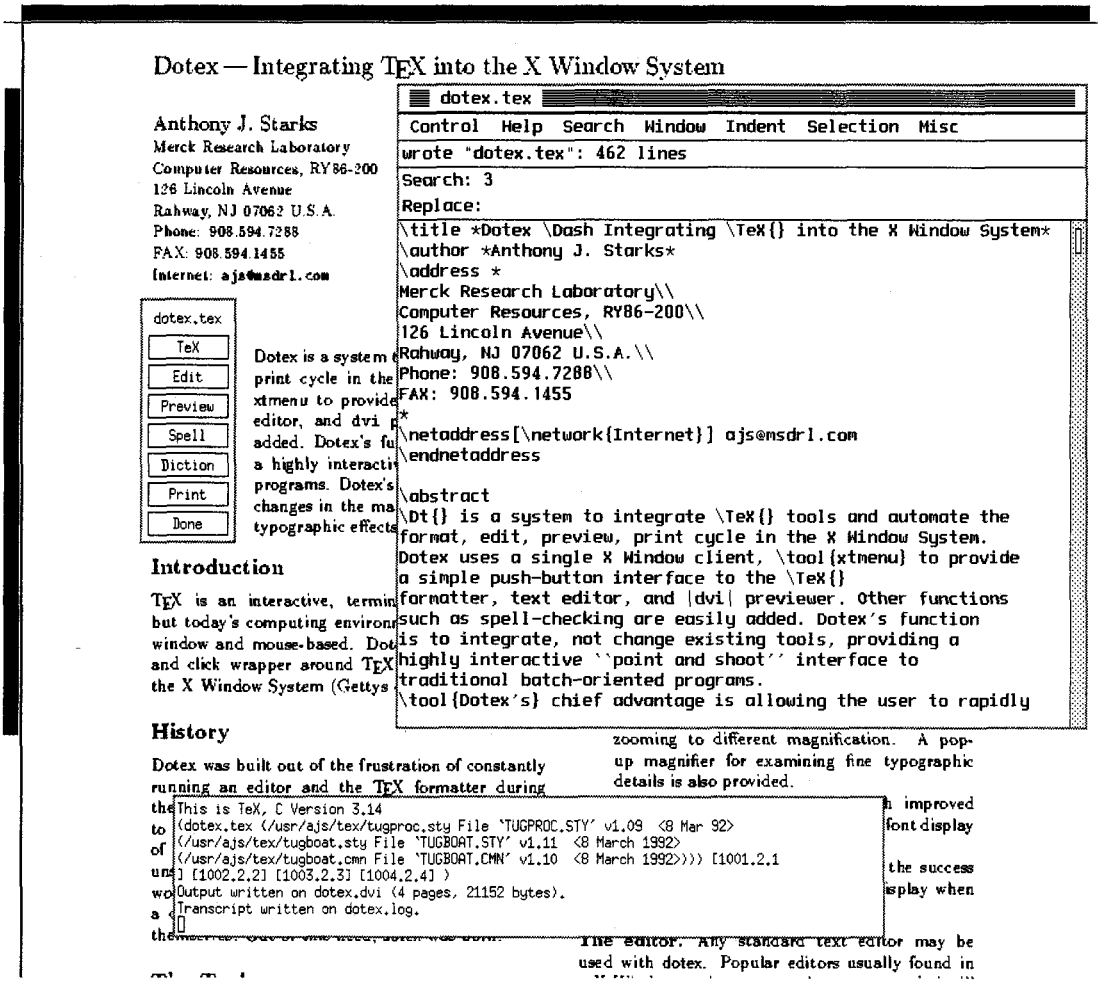


Figure 6: A `dotex` session

This typical arrangement shows the shell window at the bottom, with the edit window atop the large previewer window. The `dotex` menu is to the left of the edit window.

Appendix II — Customizing `dotex` scripts

This script is more elaborate than the simple one presented above, and adds spell and grammar checking. Note the use of horizontal orientation which is useful when there are many menu items. Also note the blank label used here to provide visual separation of the functions.

```
file=$1
xtmenu -horizontal -noquit -stdin <<!
"$file.tex" !label
TeX         %tex $file.tex
Edit        %mx $file.tex
Preview     %xdvi $file.dvi&
""          !label
Spell       %xterm -e ispell $file.tex
Diction     %striptex $file.tex | diction
Print       %dvips $file
Done        !exit
!
```

Of course `dotex` is not limited to plain TEX; this example runs LATEX and adds special BIBTEX support. Also added is an additional button to view the log file.

```
file=$1
xtmenu -noquit -stdin <<!
"$file.tex" !label
LaTeX       %latex $file.tex
BibTeX      %bibtex $file
Clean       %rm -f $file.aux $file.bbl && echo Work files cleaned.
Edit        %mx $file.tex
Log         %mx $file.log
Preview     %xdvi $file.dvi&
Print       %dvips $file
Done        !exit
!
```

This script is used to automate the creation of documents with included figures. Buttons are defined to popup the figure drawing tool `xfig` and to translate the figure file to LATEX.

```
file=$1
xtmenu -horizontal -noquit -stdin <<!
"$file.tex" !label
LaTeX       %latex $file.tex
Edit        %mx $file.tex
Log         %xterm -e less $file.log
Figure      %xfig $file.fig&
fig->TeX    %fig2dev -Llatex $file.fig >${file}-fig.tex
Preview     %xdvi $file.dvi&
Print       %dvips $file
Done        !exit
!
```

Another way to run **xtmenu** is with multiple *menu files*, each tailored to a different formatting situation. In this case, the **dotex** shell script might look like this:

```
#
#  dotex -- automate TeX and tools
#  (multiple menu file version)
#
#  Usage:  dotex [-l] [-p] [-b] file
#
mdir=/usr/local/lib/xtmenus
option=$1
TeXfile=$2
export TeXfile
case $option in
-l)     xtmenu -noquit -m $mdir/latex.xtm&;;
-p)     xtmenu -noquit -m $mdir/pictex.xtm&;;
-b)     xtmenu -noquit -m $mdir/bibtex.xtm&;;
*)      xtmenu -noquit -m $mdir/plain.xtm&;;
esac
```

Figure 7: A **dotex** script with multiple menu files

where the different menu files are stored in a special directory **/usr/local/lib/xtmenus**. All of the menu files use a common variable, **$TeXfile** when referring to the TEX source file. The different styles of TEX use are invoked by using a different option character, as in:

```
dotex -l file
```

to run the LATEX specific setup.

Appendix III — The newer program

```
/*
 * newer -- compare the age of two files
 *
 *
 * Usage: newer file1 file2
 *
 * A zero status is returned
 * if and only if file1 was modified after file2.
 *
 *
 *
 */

#include <sys/types.h>
#include <sys/stat.h>

main(argc, argv)
        char            **argv;
        int               argc;
{
        struct stat     s1, s2;

        if (argc != 3)
                exit(1);

        if (stat(argv[1], &s1) < 0)
                exit(1);

        if (stat(argv[2], &s2) < 0)
                exit(1);

        if (s1.st_mtime >= s2.st_mtime)
                exit(0);
        else
                exit(1);
}
```

GNU Emacs as a Front End to LaTeX

Kresten Krab Thorup
Dept. of Mathematics and Computer Science
Institute of Electronic Systems
Aalborg University
DK-9220 Aalborg Ø
Denmark
krab@iesd.auc.dk

Abstract

As LaTeX and TeX are more widely used, the need for high-level front ends becomes urgent. This talk describes AUC TeX, one such high-level front end for LaTeX, written for GNU Emacs.

AUC TeX does not at all attempt to be WYSIWYG; rather, it offers the author of a LaTeX document a plain ASCII file, together with a number of features to simplify and help the author perform certain tasks such as running (LaTeX), finding errors, and simply typing in the document.

Background — The GNU Emacs Paradigm

GNU Emacs is a powerful text editor which, very much like TeX, leaves the sophisticated Emacs user with the choice of affecting its behavior on demand. The extension language of GNU Emacs is a derivation of lisp called Emacs-lisp. Just as TeX allows you to associate functionality to a token, Emacs lets you bind functionality to the keys of your keyboard. When TeX implicitly inserts a `\vbox` whenever a plain letter is encountered, Emacs implicitly calls the function `self-insert-command` (which simply inserts the letter in the current buffer) whenever a key is pressed. Emacs-lisp is itself a fully featured general-purpose language; this makes it possible to make it behave anyway you want.

At my site, Emacs is being used for many different applications, from various kinds of text editing (such as source code) to Mail and Usenet news reading. The low-level editing features of Emacs will stay the same regardless of the current application. As a user becomes familiar with the basics of the editor, he will have only a few problems converting to a new application.

As many users are used to (and pleased with) Emacs, it is desirable to use Emacs for (LaTeX) editing too, and this is where AUC TeX enters at the scene. AUC TeX is a general customizable environment for editing LaTeX documents. It is written entirely in Emacs-lisp, which makes it (relatively) easy to modify to suit personal taste and needs.

Advantages of Structure-Oriented Editing

AUC TeX is an application for editing (LaTeX) documents, especially LaTeX. The most general advantage is that by knowing the general structure of a LaTeX document, which is quite simple, AUC TeX can help a user perform certain tasks. The following is an outline of the major features of AUC TeX.

- Insertion of templates for logical-structural compositions such as environments and sections.
- Hot-keys for easy access to certain often used constructs, e.g., font changes, accented letters, and mathematical symbols.
- Running application programs (such as TeX), and then parsing the output so that errors in the document may be located easily.
- Support for multi-file documents.
- Online help for (LaTeX) error messages.
- Outlining — i.e., manipulating the document as a composition of nested/sequential logical constructs.
- Instant formatting and indentation of the ASCII-document in order to make it easier to read.
- 'Completion' (and thereby spell-checking) of partially written control sequences.

AUC TeX incorporates a large number of well-known facilities for user interfacing. At first glance, it may seem that it's just too much, but it has been put together in such a way that you can easily use just parts of it, without even knowing about the rest. Though featuring a lot of fancy functionality, AUC

TeX still conforms to the standard Emacs environment — basic operations such as cursor movement and file handling are the same.

Many of the features of AUC TeX which are not basic Emacs functionality are implemented conforming to certain unwritten conventions, so that if you have already tried some other Emacs mode, such as C-mode for instance, you will simply know what do do.

Writing a LaTeX Document

I guess it is about time to let you know how it really works. We will now go on a little journey through the world of AUC TeXand explore some of its features.

Getting started. To start AUC TeX you simply run Emacs, with a LaTeX file as argument:

```
prompt$ emacs myfile.tex
```

Emacs will start and enter LaTeX mode. If you have already started Emacs, you may enter LaTeX mode, by typing `M-x latex-mode`.[1]

The first thing you are likely to do is to insert a template for the overall document structure. To do this, press `C-c C-c`. You will be prompted to insert an environment. Since the document is empty so far, AUC TeX will choose `document` as the default environment. Now type `RET` and you will be prompted for a document style, which defaults to `article`. Typing `RET` once more will prompt you for a list of style options. Write something and type `RET` again. Now AUC TeX will display a template something like the following on your screen:

```
\documentstyle[a4,12pt,dk]{article}

\begin{document}
_
\end{document}
```

and the cursor will be placed at the _.

To insert some sectioning command, press `C-c C-x`, and you will be prompted for a command (section, chapter, etc.), a title, and a label for it. Again, AUC TeX will look at the document so far, and choose some appropriate default for the command, in this case `section`.

[1] Emacs keying sequences are usually a combination of `Control` + another key, or `Meta` + another key. Thus, the notation `C-x` means "while holding down the `Control` key, type the `x` key"; `M-x` means "press and release the `Meta` key (which can be system-dependent) and then type the `x`." Combinations of sequences (such as `C-c C-x`), or combinations of sequences + explicit words (such as `Meta-x latex-mode`) are also possible.

In general, environments are inserted with the `C-c C-c` sequence. Some of the environments have special handlers attached to them: if you are inserting a figure environment, it will ask for placement modifiers, label, caption and whether the figure should be centered or not.

Completion. Since you have now specified the document style and options, AUC TeX is now (in principle) aware of all the commands you may use in this particular document. One of the AUC TeX advantages is to allow *completion* in various situations. To try this, type `C-c C-c` again, and press `TAB`. AUC TeX will now display a list of all the environments you can possibly use in the current document. If, for example, you want to insert a verbatim environment, just type `ver TAB`, and AUC TeX will complete the word `verbatim` for you. In case more environments start with the sequence `ver`, it will complete as much as it unambiguously can, and display a new list of possible completions.

Another, more general application of completion is the completion of control sequences. Type a part of some command, and press `ESC TAB`. If you want to insert the command `\thispagestyle`, which is very long and tedious to type in, especially since you are likely to introduce an error, you can simply type

```
\this ESC TAB
```

and AUC TeX will complete the entire command `\thispagestyle` for you. As before, a list of possible completions will be displayed in case of ambiguity.

Invoking LaTeX

Now suppose you'd like to process the contents of the buffer, i.e., run the file through LaTeX. This is handled very easily from within AUC TeX. Press `C-c C-a` (Mnemonic: do it *all*), and the current view will be split in two, of which the lower half is used for TeX output, while you can still edit the document in the upper half.

AUC TeX also allows you to process only part of a document. This is done by marking the region you'd like to have processed, and pressing `C-c C-d` (Mnemonic: *don't* try to remember it). A temporary file to be processed by (LaTeX will then be created in the current directory, in which AUC TeX will put the preamble there (i.e., from `\documentstyle` to `\begin{document}`), after the marked region, and then insert an `\end{document}` in the bottom.

Multi-file documents. In case your document is spread over several files, AUC TeX can handle that too. If you insert the sequence:

```
%% Master: somefile.tex
```

then the file `somefile.tex` will be the file actually to be formatted if you invoke `C-c C-a`. Also, if you invoke `C-c C-d` to format just a part of the document, then the preamble will be sought in that file.

Another mechanism is also provided. If you have neither specified a `%% Master` line, nor does your document contain a proper preamble, then AUC TEX will insert a command to load the file `texheader.tex` in the beginning of the file, which is then supposed to contain some standard preamble.

Debugging facilities. In case errors occur, the message 'errors!' is shown in the echo area, and you are asked to press `C-c C-n` (Mnemonic: *next* error) to locate the first error. Doing this will place the cursor as close as possible to the first reported error, and a description of possible causes of the error is displayed in the lower half of the view.[2] This may be repeated as often as there are more reported errors. Please note that one error is likely to produce more, so if you don't understand what some error message means, it may be a good idea to reprocess it all, to see if your changes have perhaps eliminated some errors.

Locating the error To find some error, AUC TEX parses the log file. This is perhaps one of the most interesting parts of AUC TEX. The parsing is based on the fact that whenever TEX encounters an error, it will print something like the following sequence to the log file:

```
! Something's wrong--perhaps a missing \item.
  ⟨context lines⟩
l.234 \section
                {Formatting}
```

This means that the error "Something's wrong–perhaps a missing `\item`" occurred at the control sequence `\section` of line 234 of the current buffer. This spot is quite easily located — and this is where the interesting part begins. Whenever TEX reads a file, it will print some sequence like the following to the log file:

```
(somefile.tex [3] [4]
  ⟨other log messages⟩
)
```

The parsing of this construct is complicated somewhat by the fact that ⟨other log messages⟩ may actually be any arbitrary text, and especially parentheses, which may be unbalanced, and perhaps even followed by things that may look like file names.

[2] Leslie Lamport kindly supplied me with the "help" texts for (LA)TEX error messages, as descibed in Lamport (1986)

The Big Picture

Several features of AUC TEX are aimed at making it easy to overview your document. This can be a great help, especially if you must edit documents not written by yourself. In this talk, I will describe the features for formatting, and outlining.

Formatting When you write a document using AUC TEX, you will notice that the text is automatically formatted and indented as you write it. Lines are automatically wrapped at a particular column, and the left margin is also adjusted to reflect the structure of the document.

To get an idea of what this formatting stuff means, here is a sample of the 'item list' from the beginning of this article as it appears in the document.[3]

```
\begin{itemize}
\item Insertion of templates, for
  logical-structural compositions such as
  environments and sections.
\item Hot-keys for easy access to
  certain often used constructs, e.g.,
  font changes, accented letters, and
  mathematical symbols.
\item Running application programs
  (such as \TeX), and then parsing the
  output so that errors in
  the document may be located easily.
  ...
\item 'Completion' (and thereby
  spell-checking) of partially written
  control sequences.
\end{itemize}
```

There are several advantages in such a formatting scheme. Most important, it is easy to locate a given point in the document, as the ASCII-text reflects the actual printed document. Moreover, the indentation is a great help in localizing errors in the document — if the indentation doesn't look right, you've probably missed some closing construct, such as an `\end` tag.

There are many aspects of formatting in AUC TEX. First of all, instant processing is automatically taking place, while you write a document. Next, reformatting of paragraphs is very useful to clean up a some messy construct, and this even works for things like an item of a list. Last, general reformatting features are available, which let you reformat sections, environments or the entire document. Refer to the

[3] The verbatim sample shown here is formatted with a narrower margin than in the actual document in order to fit the column.

function listing in the appendix of this article for further information.

Outlining. A special minor mode is available along with AUC TeX to allow outlining of a LaTeX document. The outlining feature allows body text or subheadings to be made temporarily invisible or visible again. Such invisible text is attached to the end of the heading to which it belongs, and moves along with it. A heading under which some body text is hidden is marked with an ellipsis (...). For example, the current section looks like this, when totally collapsed:

```
\section{The Big Picture} ...
\subsection{Formatting} ...
\subsection{Outlining} ...
```

This outline mode is enabled via the command `M-x outline-minor-mode`, after which certain key sequences can be issued to manipulate structural elements of the document. See Thorup (1992) for further documentation.

Other Subtle Features

Mathematical symbols. A special minor mode is available for easy access to mathematical symbols, which is often convenient when writing an application full of them. The general idea is that once you've entered this mode, pressing the sequence '⟨left quote⟩-⟨letter⟩' causes some symbol to be inserted, e.g., `'-a` inserts `\alpha`, `'-b` inserts `\beta`, etc. The translation is controlled by a table, which may be easily redefined if needed.

Accented letters. As with mathematical symbols, there is another a minor mode for entering accented letters with the key sequence '⟨accent⟩-⟨letter⟩'. The mapping is easily redefined by the user.

Availability

AUC TeX is available by anonymous ftp to the address `iesd.auc.dk`, but should also be available at major TeX archives around the world.

If you do not have ftp access, you may send mail to `auc-tex_mgr@iesd.auc.dk` who will be happy to mail you a copy of the latest release.

A version of AUC TeX for Freemacs (Emacs for the IBM PC), written by Richard Flamsholt Sørensen (`richard@iesd.auc.dk`) is also available as part of the Freemacs distribution.

Acknowledgements

I should like to thank Per Abrahamsen, Lars P. Fischer and a lot of unnamed people on the net, for contributing to the discussion/development of AUC TeX; ICL Data Denmark for sponsoring my travel to the Annual Meeting; and finally Leslie Lamport, who supplied me with the help text for LaTeX error messages.

Bibliography

Lamport, Leslie. *LaTeX, A Document Preparation System.* Reading, Mass: Addison-Wesley, 1986.

Stallman, Richard M. *The GNU Emacs Lisp Reference Manual.* The Free Software Foundation, 1992.

Thorup, Kresten Krab. *The AUC TeX Reference Manual.* To appear, 1992.

Functional summary for AUC TEX version 5.6

Run TEX/LATEX on buffer	C-c C-a
Run TEX/LATEX on region	C-c C-d
Print the DVI file	C-c !
Preview dvi file	C-c C-p
Next error in TEX/LATEX session	C-c C-n
Run BibTEX on buffer	C-c @
Run makeindex on buffer	C-c #
Kill job	C-c C-k
Re center output buffer	C-c C-l
Toggle Debug Boxes	C-c C-w
Home Buffer	C-c C-h
Insert **bold** syntax	C-c C-b
Insert *italics* syntax	C-c C-i
Insert roman syntax	C-c C-r
Insert *emphasized* syntax	C-c C-e
Insert `typewriter` syntax	C-c C-t
Insert *slanted* syntax	C-c C-s
Insert SMALL CAPS syntax	C-c C-y
Insert Sectioning command	C-c C-x
Insert `\begin{}...\end{}` environment	C-c C-c
Comment out a region	C-c ;
Comment out a paragraph	C-c '
Un-comment a region	C-c :
Un-comment	C-c "
Insert item	M-RET
Format a paragraph	M-q
Format a region	M-g
Mark a section	M-C-x
Format a section	M-s
Mark an Environment	M-C-e
Close off an Environment	C-c C-f
Format an Environment	M-C-q
Complete Symbol	M-TAB
Up-list	M-}
Terminate Paragraph	RET
Smart "Quote" Insert	"
LATEX-indent-line	TAB
Re-indent, then newline and indent	LFD
Terminate paragraph	C-c LFD

Writing Reports with More than a Hundred People

Walter van der Laan and Johannes Braams
PTT Research Neher Laboratories
P.O. Box 421
2260 AK Leidschendam
The Netherlands
W.vanderLaan@research.ptt.nl; J.L.Braams@research.ptt.nl

Abstract

This paper describes a system that produces project status reports using LaTeX. The reports contain both textual and financial information. The textual part of the status reports is written by over a hundred people who don't need to know what LaTeX is. The financial information is retrieved from a database.

Introduction

Each quarter of a year the clients of our research center receive status reports concerning all of their projects. As you might expect these reports are typeset using LaTeX. What might be more of a surprise is the fact that the status reports are written by over a hundred project managers who don't need to have any knowledge of LaTeX. They write their status reports at different sites, using various computer systems, word processors and editors.

The first section gives an impression of the environment and history of this reporting system. The second explains the least that a project manager needs to know when using the system. A mail server is used to collect all project status reports. This server is discussed in the third section. The fourth section describes the generation of reports. Some general conclusions are listed in the last section.

Environment and History

With about 95,000 employees, PTT is the largest company in the Netherlands. The business of PTT is selling postal and telecommunication services. The reporting system described in this paper was made for PTT Research, a division of PTT with about 800 employees. PTT Research does most of its work under contract to other PTT divisions; typically there about 350 research projects for about 30 PTT divisions.

Each quarter, all project managers write a one-page status report to keep their client informed. These status reports consist of the following four sections:

- a short description of the project,
- the targets for the reporting period,
- the work realized in the reporting period, and
- the targets for the next period.

For each project the text written by the project manager is pasted into a form supplied by our financial department. This form shows information from a financial package, e.g.:

- the names of the client, project and project manager,
- important dates related to the project,
- the amount of money spent so far, and
- the budget.

The completed forms are bundled and presented to our clients.

The production of these status reports used to be manual. The texts supplied by the project managers, showing all kinds of fonts and printing qualities, were literally pasted onto the form using scissors and glue. Throughout our company, about a dozen people used to be busy collecting status reports and completing forms. It took more than a month before we were able to present the status reports to our clients. During this production process it was almost impossible for a project manager to make any corrections.

The system described in this paper offers much more flexibility. People, both with and without any LaTeX experience, send their status reports to a mail server. A report generator is used to produce LaTeX files containing a mix of information from the financial database and the status reports that have been received through the mail server. It enables us to present a uniform and beautiful report to our clients about ten days after the financial closure of a quarter year. Within these ten days, project managers and their managers get several chances to correct the status reports.

Writing Status Reports

The status reports can be written as plain ASCII text or as LaTeX text. The first option is default and foolproof, it protects a project manager from any LaTeX errors. While this option is easy to use it also leaves the writer with only a few of the expressions available in LaTeX, namely paragraphs and some special characters common in the Dutch language. This has proved to be sufficient for 99 percent of the reports but leaves a few special cases, e.g., some project managers want to put formulas in their report and others have a need for symbols used in physics. In these cases the project managers write their report as LaTeX text and they have to know how to use it. The remainder of this section explains what a project manager needs to know when using the foolproof mode, i.e., plain ASCII text.

A report sent to the mail server may contain nothing but plain ASCII characters. This is a necessary constraint because the mail server receives reports written with about twenty different kinds of word processors and editors on about five different kinds of platforms. Fortunately all word processors have an option to save a text in ASCII. This means that all project managers must know how to produce an ASCII file containing their report. They must also be aware of the limitations of ASCII; e.g., no underline, no boldface, no special characters.

Project managers must be able to send their reports to the mail server. This hasn't been a problem in our organization as everybody is connected to the local network and most people are using electronic mail.

Each report must contain a few keywords. This simple syntax is needed to

- assign a project number to each report,
- separate the four pieces of text in each report, and
- separate the report from the mail headers and footers.

The next example shows the report for project number 12345. As you can see the syntax consists of uppercase keywords with four pieces of text in between:

```
PROJECT 12345
DESCRIPTION
We are working on a project.
CURRENT TARGET
Our plan was to finish the project.
REPORT
We've had a lot of problems but the
work is almost done.
NEXT TARGET
We'll finish the project in the next
quarter year.
END
```

Many people were having problems with this combination of a textual report and a strict syntax. We eliminated this problem by extending the mail server to accept all syntax errors that occurred.

Project managers have to understand the effect of an empty line in their texts: All text is aligned on the left and right margin. An empty line causes the alignment to restart at the beginning of the next line, because texts are set without paragraph indentation.

In the Dutch language one frequently finds characters such as é, è and ï. For this reason the project managers have the option of using the sequence `backslash accent vowel` whenever they need to put an accent over a vowel.

The Mail Server

All mail sent to a dedicated network address is processed by a program, which replies to every message received. The reply consists of two parts. The first part contains success and error messages, e.g.: "`i found a report about project 12345`", or: "`i removed some control characters from your text`". The second part is a copy of the received message in which all recognized texts have been removed. If all went well the second part contains nothing but keywords and mail headers. This construction is clear even to people who aren't used to syntax errors.

In the beginning we had trouble with errors in the project number. People would erroneously send us a report for project 12435 instead of 12345. This meant one could overwrite another report by mistake. We solved this problem by saving a list of the mail addresses of the senders for each report. We accept reports from any address on the list. A report from any other address is rejected but the address is added to the list. In such cases the sender receives a message saying the report has been rejected but will be accepted if the report is sent again. This warning eliminates the typing errors in project numbers but still allows different people to send updates for the same project.

In the foolproof mode all texts must be transformed into valid LaTeX texts. This transformation is described next.

Many characters special to LaTeX have to be de-activated. We handle those characters as four separate cases:

- Double quote characters are replaced by " and ", respectively.

- A backslash is added in front of the following characters: #, $, %, &, _, { and }.
- The math characters <, > and | are placed between dollar signs.
- The expression `{\tt\char92}{}` is used to insert a backslash. Note that 92 is the ASCII value of a backslash. The same procedure is used to insert the characters ^ and ~.

The sequence `backslash accent lowercase vowel` is allowed. This sequence isn't changed by the transformation; only when the vowel is an *i* is it replaced by a dotless *ı*. This change makes the sequence very uniform and easy to understand for people not used to LaTeX.

LaTeX does a great job at automatic hyphenation. However, when words are joined together by characters such as the slash and minus, LaTeX doesn't hypenate the resulting string. This causes a lot of overfull hboxes. To solve this the transformation program looks for the sequence `letter slash` or `minus letter`. Within sequences like this the slash or minus is transformed into the parameter of the `nw` command (nw stands for *new word*), e.g.: `man/woman` is replaced by `man\nw{/}woman`. The `nw` command is expanded to let "man" and "woman" be separate words divided by a slash. The LaTeX definition is:

```
\newcommand{\nw}[1]
  {\hspace*{0pt}#1\hspace*{0pt}}
```

This transformation has proved to be sufficient in avoiding almost all overfull hboxes.

Last but not least, all characters unknown to LaTeX are removed during the transformation.

The *Lex* specification below (Lesk and Schmidt, 1975) produces a program that performs the transformation described above:

```
AN              [a-zA-Z0-9]
AC              [`'"^.=]
%%
                int dq = (0 == 1);
\"              { dq = !dq;
                  if (dq) printf ("``");
                  else    printf ("''"); }
[#$%&_{}]       { printf ("\\%s", yytext); }
[|<>]           { printf ("$%s$", yytext); }
[~\^\\]         { printf ("{\\tt\\char%d}{}",
                    yytext[0]); }
\\{AC}i         { printf ("\\%c{\\i}",
                    yytext[1]); }
\\{AC}[aeou]    { printf ("%s", yytext); }
{AN}[/-]{AN}    { printf ("%c\\nw{%c}%c",
                    yytext[0], yytext[1],
                    yytext[2]); }
\t              { putchar (' '); }
\n              { putchar ('\n'); }
.               { if ((yytext[0] >= 0x20)
                  && (yytext[0] < 0x80))
                  printf ("%s", yytext); }
```

Text sent in LaTeX mode is not transformed by the mail server. A copy of the text is transformed into a complete LaTeX document by adding a header and a footer. The mail server passes this document to the TeX compiler and afterwards checks the log file for errors. If an error occured, it rejects the text and adds the compiler messages to the reply.

The mail server program was written using *TPU*, the VAX/VMS Text Processing Utility (Digital VMS manuals, 1988). *TPU* and LaTeX make a great team and have allowed us to build this system in a short time. If you ever need a utility to process text, try *TPU*.

Report Generation

The four pieces of text per project are stored in a separate directory as four files per project. All of these texts have been tested or transformed by the mail server. These texts are ready to be typeset in any textwidth or font.

Our relational database system stores a lot of project information. This information is transformed into LaTeX strings just as the texts supplied by the project managers are transformed.

With an application program our financial department can define which projects are to be included in a report and in what sequence they are to be included. Every set of projects defined can be printed in several ways; e.g., a report containing:

- all information about a project on one page,
- only the financial information, and/or
- only the description of each project.

When we started work on automating the report generation process we first agreed upon an interface between the report generator and LaTeX that consists of a few special commands.

In Figure 1 an example of a status report is shown. At the top of the page some general information about who ordered the project and who runs it is shown. This information is repeated on a continuation page as you can see in Figure 2. Then follows a short description of the project and its targets. In this case the description is too long and is therefore continued on a second page. The next texts discuss the targets for the reporting period, the work done in the reporting period and the targets for the next reporting period. At the bottom of the page some financial information on the project is included. Both this financial information and the general information at the top of the page is extracted from the database.

For each of the text fields a LaTeX environment is defined. The task of these environments is to store the text parts in four boxes of the right size. To pass the information that is printed in the header and footer of a report we defined a few LaTeX commands with parameters. All information that belongs to a particular project should be inside yet another LaTeX environment. The task of this environment is to:

- start a new page,
- select the correct page style,
- write some information about the project to a table of contents file, and
- make sure that all information accumulated is put on the page.

Originally, the layout of the report form was defined in terms of a number of characters per line, and a number of lines for each of the four text fields. It was also specified that a report for any project should occupy not more than one page.

With typeset text, the specification of the number of characters per line is not particularly useful, because it can vary with the kind of characters that are used in the text. Also, the manual process of putting together the status reports had shown that sometimes a project manager would produce more text than would fit in the field for which it was meant.

Because the sizes of the fields are fixed, we had to choose what to do. We could typeset the portion of the text that would fit into the field and either

1. let the rest print over the next field, or
2. discard the rest, or
3. store leftover text and print it on a second page.

Option 1 clearly is unacceptable; the result would be both ugly and unreadable. Option 2 would possibly result in texts ending in a weird manner. Choosing this option would, however, force the project managers to be brief. It was finally decided to use option 3. One of the reasons for this decision was that the implementation of options 2 and 3 is almost the same.

The implementation of this part of the LaTeX style file is based on the use of the `\vsplit` command. The four environments discussed before scoop up the text and typeset it in a `\vbox` `\pickup@box` of the appropriate width. The contents of the `\pickup@box` are copied into a different box for each environment using the `\vsplit` operation. The resulting height of the `\pickup@box` is then measured. If it is not zero there is more text than fits into the field. In that case an indication that there is more text is appended to the first part of the text, and the rest is stored away to be put on a second page.

Conclusions

The system imposes no special organization upon its users. Our users write and send their reports using the computer system that they use for their everyday work. In some departments the reports are gathered by one person who sends all reports and report updates to the mail server. In other departments all project managers send and update their own reports. The nice thing about a mail server is that it doesn't mind where a report was made or who made it.

The two modes, LaTeX or foolproof, have made the system both very flexible and very easy to use. No special training is required for people who use the system in the default foolproof mode. People with special requirements, on the other hand, are very happy with the LaTeX mode.

LaTeX separates the contents of a document from the form in which it is presented. This separation was of great benefit to us during the development of the system. It allowed us to make a very clean and easy division of the work to be done. After defining the different styles needed for our system one of us would work on the generation of LaTeX reports using the defined styles, while the other would work on the creation of the styles.

At first we implemented the reports to be generated interactively. This didn't work very well because LaTeX consumes large amounts of cpu time. At the moment the production of reports is implemented as a lower priority background process.

Bibliography

Lesk, M.E., and E. Schmidt. "Lex — A Lexical Analyser Generator." Bell Laboratories. Murray Hill, New Jersey, October 1975.

VAX Text Processing Utility Manual. Digital VMS Manuals, **5B**, April 1988.

PTT Research Kwartaal Rapportage

92/01

Projectnaam	PART. WERKZAAMHEDEN SC18	Projectnummer	61006
Opdrachtgever	RAAD VAN BESTUUR RVB	Onderdeel/hafd	TI INFORMATIEDIENSTEN
Afdeling	RVB	Hoofd	TERPSTRA, A.P.
Hoofd	J. WAGE (secretaris LBR)	Projectleider	W. REMMERS
Contactpersoon	W.J. THIEME	Begin/Eind datum	10192 / 311292
Kenmerk	OVK D.D. 15/07/1990	Kenmerk offerte	91-A041

Korte omschrijving en doel van het project

Binnen ISO/IEC JTC1 is SC 18 gericht op de standaardisatie-aspecten van creëren en manipuleren van documenten. SC18 omvat de volgende gebieden:
* document architecturen en semantiek (ODA, SGML, DSSSL)
* document interchange formaten (ODIF, ODL, SDIF, DTAM)
* basis inhoudsarchitecturen (o.a. teken, raster, muziek) en weergave-aspecten (kleur, fonts)
* hypermedia (HyTime, SMDL)
* printeraansturing (SPDL, DPA)
* message handling en message store (X.400 of MOTIS)
* document filing & retrieval (DFR)
* *User System Interfacing (keyboard layouts, Icons & Symbols, dialoog standaarden)*

Zie volgende pagina

Gesteld doel voor dit kwartaal

Gezien het belang van de SC18-onderwerpen, de breedte van het aandachtsgebied en de doelstelling de werkmaatschappijen meer bij dit werk te betrekken, dient er een duidelijk plan van aanpak geformuleerd te worden. Vanwege een wisseling in de bezetting is deze aktiviteit vorig jaar opgeschort. Hieraan zal binnenkort wel veel aandacht besteed worden.

Korte rapportage

Op dit moment zijn een groot aantal standaarden en/of uitbreidingen op de basis-standaard (zie boven) in bijna-finale vorm. Tegelijkertijd blijken er nog grote definitie-problemen te bestaan, en blijkt tussen verschillende standaarden grote overlap te bestaan. In vorige rapportages is al genoemd de relatie tussen 'character, graphic character, font en glyph', Message Store Extensions versus DFR en DTAM, ODA en SGML. tegelijkertijd worden er in hoog tempo nieuwe werk-items voorgesteld. De wens naar stabiele standaarden is momenteel strijdig met de momenteel gehanteerde ISO-planning en het aflopen van de studie-periode van CCITT. De voorjaarsvergadering is bij OC\'E gehouden, waar een interessante presentatie over de geschiedenis van het bedrijf en de bemoeienis met ODA werd gegeven.

Gesteld doel voor het volgend kwartaal

1. Het aktief volgen van ontwikkelingen binnen SC18.
2. Afronden plan van aanpak (tweede kwartaal 1992)

Inzet S&O	afgelopen kwartaal	t/m afgelopen kwartaal	overeenkomst	perc. gerealiseerd
Aantal Uren	81,00	81,00	320,00	25,30
Bedrag (fl)	16.038,00	16.038,00	63.360,00	25,30

1 Uitsluitend voor gebruik binnen PTT

Figure 1: A sample report for a project

However, when it came to the use of the `picture` environment, I feared that I might have met my match.

Flavors of TEX at the SSC. There are many flavors to TEX in use throughout the SSC Laboratory. These flavors can be compared to the 31 flavors offered by Baskin-Robbins; however, there are only 5 TEX flavors.

At the SSCL, there are the three traditional TEX packages — TEX, LATEX, and AMS-TEX; but there are two others. Since the SSC is a high-energy physics (HEP)-type environment, we also use PHYZZX (a TEX macro package containing many HEP formatting requirements) and TEXsis (another HEP-type TEX macro package). PHYZZX was developed at the Stanford Linear Accelerator Center by M. Weinstein; TEXsis was developed as the result of a TechRpt format described by W. Groppe and from TEXsis 1.01, the TEX thesis format by E. Myers; with modifications made by F. Paige.

The use of TEX at the SSCL provides a multitude of learning experiences. Familiarity with each package is required to become the TEX support person at the laboratory; but more than that, it keeps one fluid in the use of every form of TEX.

Graphics experience: Past and present. My first attempt at graphics saw the first 16 hours spent learning the limitations of the `picture` environment. However, this was not the only problem I encountered. Also, I had to master the limitations of the Imagen LBP-10 laser printer. One project that I attempted was drawing a simple piece of graph paper for plotting all my pictures (Figure 2). The ability to produce complex mathematical diagrams in fields such as combinatorics, graph theory, discrete and applied mathematics was soon achieved (Figure 3).

Working at the Superconducting Super Collider Laboratory (SSCL) since 1989, my graphics experience blossomed when I used a Macintosh with its enhanced graphics capabilities. This new experience led to including graphics into LATEX documents. However, graphics inclusion does not stop with a graphics package; it also sees the inclusion of Macintosh or DECstation screen dumps (or captures) into these documents. According to Freedman (page 612), a *screen dump* is "the ability to print the entire contents of the current display screen".

Still using the Macintosh for most of the graphics placed in documents, a DECstation 2100 is used for screen dumps of workstation windows, applications, and environments. Being able to locate graphics applications that allow PostScript (PS) or Encapsulated PostScript (EPS) conversion has been interesting.

This experience opened an opportunity to produce completely professional-looking documents.

Documents With Graphics Included

In preparing several users' guides requiring the graphics inclusion, an investigation was started on the use of the `\special` command. The `\special` macro is used for inserting illustrations or graphics in text. This led to some interesting discoveries about printer drivers, available commercial printer packages, the PostScript world, and seeing completed documents with self-contained graphics. Results from these discoveries have led to the production of three documents — the SSCL Computer User's Guide, the SSCL Computer Operations Manual, and the SSCL Physics Detector Simulation Facility (PDSF) User's Guide and Training Manuals. The self-contained graphics in these documents were created using Deneba Canvas for a Macintosh, Unigraphics CAD software from an Intergraph workstation, and screen captures from the Macintosh and a DECstation 2100.

Details of the procedures used for the graphics inclusion and the DVI-to-PS packages used are discussed later in this paper. A little history is now given on the above documents.

The SSCL Computer User's Guide and SSCL Computer Operations Manual were created with graphics inclusion. Only one figure in the first versions of these two documents was non-contained. Graphics packages used for these documents were Deneba Canvas and Unigraphics CAD graphics converted to Canvas-readable format. These graphics were next converted to EPS format and transferred to the VMS mainframe. After transferring the converted graphics to the mainframe, Northlake Software's T2/script product was used for the `\special` command inclusion.

These documents are in the second revision stage and are being done on a DECstation 2100 workstation running the Ultrix operating system. Some of these graphics are carryovers from the first versions, but the one non-contained figure is being replaced by an electronic version (Figure 4). Arbortext's DVILASER/PS software is used for the `\special` command inclusion.

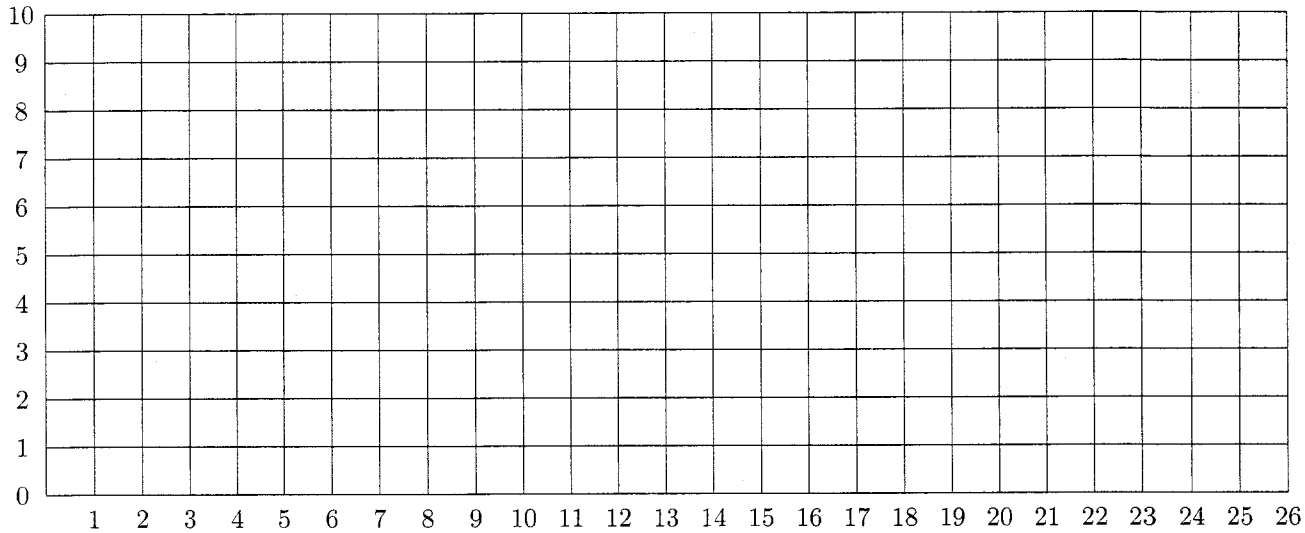


Figure 2: First project — creating graph paper

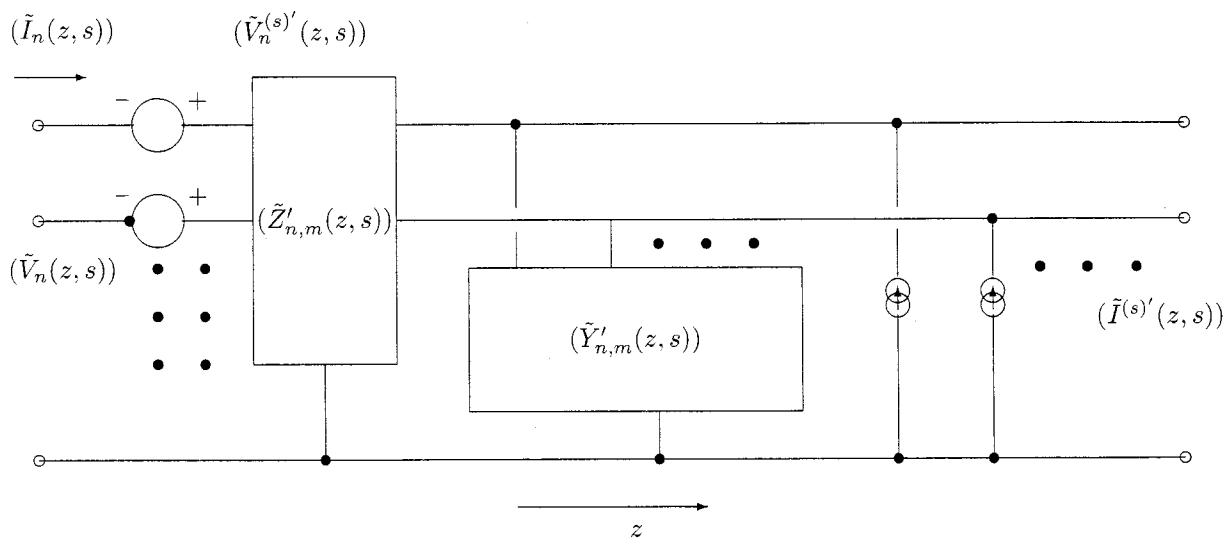


Figure 3: Second project — schematic diagram

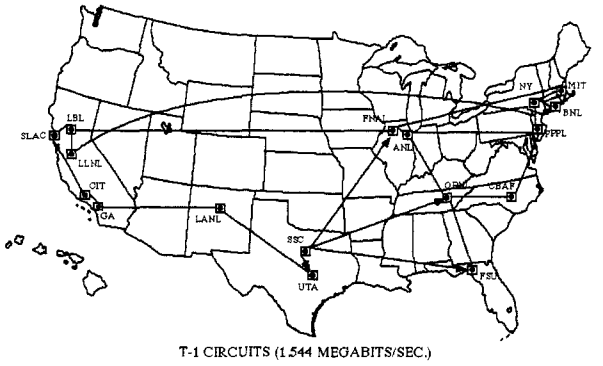


Figure 4: Non-contained figure now included

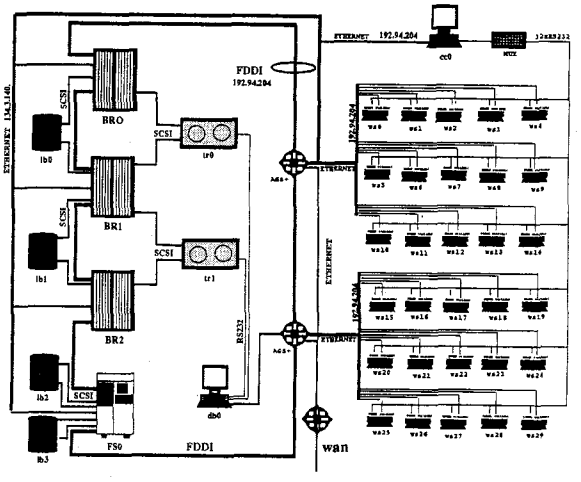


Figure 5: PDSF computer system

The PDSF User's Guide and Training Manuals were written to support users of the Physics Detector Simulation Facility, a complex system that enhances the scientists ability to do high level simulations of detector experiments. The graphics inclusion and DVI-to-PS packages used were the same as the other documents, except for one difference. One figure was created using a software package called NetCentral Station (Figure 5).

After all this background history, you are probably wondering how do you create graphics. What packages are used? How do I import/export them into PostScript or EPS format and include them in your LaTeX document.

Graphics Packages Used

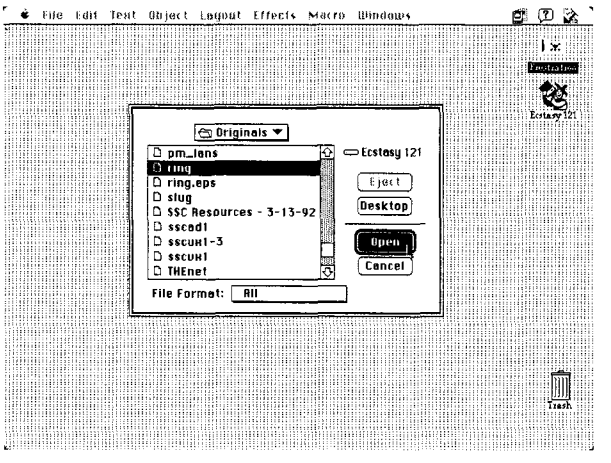
Schwer (page 195) provided the basic steps needed for including Macintosh graphics in LaTeX documents. These steps are:

- To create graphics with the user's favorite Macintosh application;
- convert graphics into PostScript representation;
- transfer the PostScript file to the LaTeX host machine; and
- include the PostScript file in the LaTeX document via the DVI-to-PS driver's `\special` command.

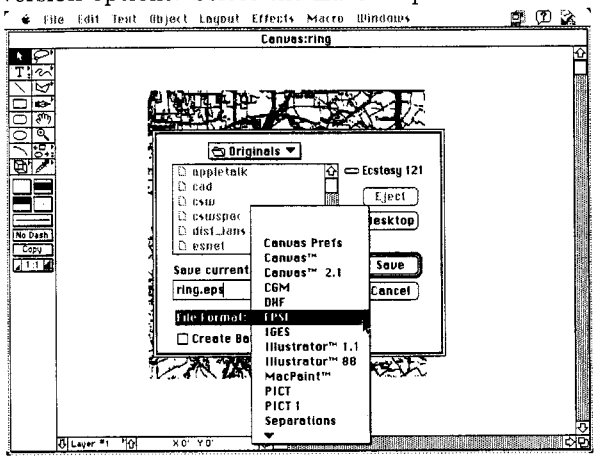
The next four sections describe the methods of creating the PS representation of the graphics into a usable file for the `\special` command.

Deneba Canvas 3.0 is a complete graphics package created by Deneba for the Macintosh. By complete, it is meant that Canvas combines MacDraw II and MacPaint 2.0 to create a great graphics image. Canvas 3.0 allows the conversion of Canvas files into several selections, such as EPSF, Adobe Illustrator 1.1, etc. Once the conversion is done, the new files are in EPS representation, thus having the ability of being opened, modified, and saved into other Macintosh programs, or imported into different Macintosh programs. However, the purpose of this paper is to show how to include graphics into LaTeX documents. Therefore, the following procedures, shown pictorially below, explain how to convert the file into EPSF format.

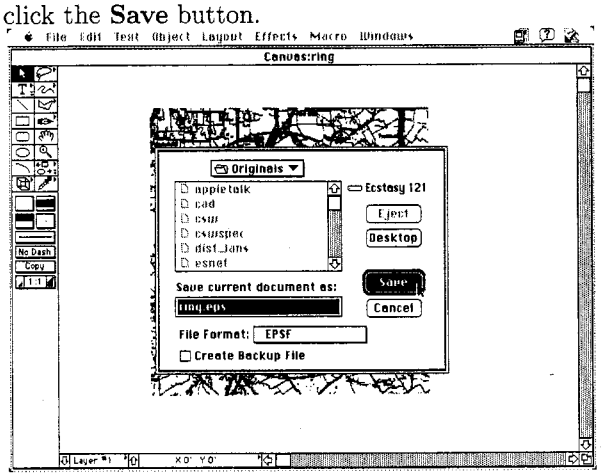
- Open the file to be converted, then
- Click on the **Save As...** button providing a filename, such as *filename.eps*.



Another dialog box appears allowing selection of the proper **File Format** with all available conversion options. Select the **EPSF** option.



After choosing the appropriate conversion format, click the **Save** button.



The next step is transferring the **.eps** files to the host computer. This procedure is not covered in the scope of this paper.

The Unigraphics CAD graphics system controls production of engineering drawings, floor plans, etc. For the user documents produced at the SSCL, the CAD Group converts the floor plans into PICT files that are imported into Canvas. Once in Canvas, the floor plans are modified and then converted to an EPSF-formatted file.

Macintosh/DECstation screen captures. A screen capture is similar to the screen dump. The major difference is that the DECstation software allows you to capture, or take a *snapshot*, a portion of the screen versus the entire screen. Digital Equipment Corporation refers to this as their Print Screen feature. Print Screen lets you print the snapshot immediately or capture the image and save it in a PS file to be printed later.

A Macintosh screen dump is done by pressing the **3** key with the **Apple** and **Shift** keys held down. This results in a MacPaint file being created containing the current screen regardless what type of application is running.

Canvas is used to open the MacPaint file. Once the file is open, rotation and sizing modifications are made. This file is then converted to the appropriate EPSF-formatted file.

The NetCentral Station (NCS) is Cisco Systems' network management product that is designed to monitor complex internetworks and to simplify in-depth network planning and analysis. NCS incorporates a high-level graphics editor that provides users with the tools to create and position their own graphic images of routers, bridgers, hosts, and links. The graphics editor is able to convert active, online graphics into suitable raw PS files that are placed directly into LaTeX documents.

The ability of seeing the printed page becomes an exciting reality. Thus, the next step was to acquire the right DVI-to-PS printer drivers.

How DVI-to-PS Printer Packages and `\special` Commands Intertwine?

Schwer (page 195) states two key points in his article:

1. There are several DVI-PostScript drivers available and they all treat the `\special` command differently.
2. Not all PostScript devices are the same. Macintosh QuickDraw, a PostScript language shorthand, in combination with various PostScript implementations of DVI drivers produces different results on different PostScript drivers.

These observations are certainly true. Most users are not aware that the way they use the `\special` for figure inclusion is not standard. The reason for this is that a local TeX wizard has located specific macros to work with the Laboratory's `dvi` processing programs. Thus, there is no reason for users to look under the sheets to discover that not all `\special` commands work the same way.

At the SSCL, there are three different computing platforms available — VMS, UNIX, and Macintosh (with PCs soon to be added) — it became necessary to learn how the `\special` command worked on these platforms. Referring back to Schwer's two key points, I discovered that the `\special` command

did work differently depending on which computing platform and DVI-to-PS packages were used.

Discussion is presented about three commercial packages currently in use at the Laboratory:

- VMS: Northlake Software's T2/script
- UNIX: Arbortext's DVILASER/PS
- Macintosh: Blue Sky Research's *Textures*

Each of these packages support capabilities other than figure inclusion (e.g., selecting color for text; raw PS code insertion; landscape mode; and TIFF, PCX or PICT file insertion).

VMS. Northlake's T2/script (1989) software translates "TEX DVI files and fonts into PostScript, for printing on many different PostScript printers... PostScript commands and Encapsulated PostScript files may be inserted in the job using TEX `\special` commands."

The use of this package was easy. Steps for using this package are presented below.

1. Place the `\special` command where you want the graphics to appear using the syntax:

   `\special{insert filename.eps [qualifier]}`

2. Four additional qualifiers are available for use. They are:

`orientation=`± 1	Rotates picture 90 degrees
`magnification=`*n*	Enlarges or reduces graphics (default = 1000)
`left=[`$\pm$*len*`]`	Controls left side of graphics
`top=[`$\pm$*len*`]`	Controls top side of graphics

Three examples on the use of this `\special` command are:

```
\special{insert filename.eps, left=-xxbp}
\special{insert filename.eps, orientation=-1,
    left=-180bp, top=-72bp}
\special{insert filename.eps,
    magnification=750}
```

where *bp* stands for big points (or 72 bp = 1 inch).

UNIX. Arbortext's (page 3) DVILASER/PS product translates "TEX **dvi** files into PostScript; these PostScript files can be printed on any printer or typesetter that supports the PS language". The `\special` command provides graphics inclusion into LATEX documents and provide the capability to send PS commands directly to the printer.

Arbortext's `\special` command provides the ability of embedding graphics files in the **.ps** file with one of three commands — ps: epsfile, ps: plotfile, and ps: overlay. For purposes of this paper, the ps: epsfile command is discussed. The other two are left for discovery on your own.

The **epsfile** command takes the form:

```
\special{ps: epsfile filename.eps magnification}
```

showing that the file `filename.eps` is to be inserted with the lower left-hand corner of the figure's bounding box placed at TEX's current point, scaled by magnification/1000. Magnification is optional; however, it must be an integer. Other uses of DVILASER/PS involves inclusion of extensive verbatim PS code.

Macintosh. Blue Sky Research's *Textures* is simply TEX for the Macintosh computer. The best way of describing Textures' concept of graphics inclusion is to use Blue Sky Research's (back cover, 1990 *TUGboat*) own words.

> "... Adobe Illustrator *does* do pictures, with a line quality finer than any technical pen. Or use MacDraw from Claris for technical drawings; learn it in less than one hour. Image Studio from Letraset does halftones, hand-painted or scanned. All world-class programs, all only on the Macintosh. With these tools (and many others), *Textures* does pictures—on screen, on paper, beautifully."

To begin using the *Textures* `\special` command, the following definition should be placed in your preamble.

```
\def\picture #1 by #1 (#3){
  \vbox to #2{
    \hrule width #1 height 0pt depth 0pt
    \vfill
    \special{picture #3}}}
```

After having entered this definition, the proper syntax of including an EPS or PS file is `\special{picture` *name*`}`. If the picture needs to be centered within the page dimensions, type:

```
\centerline{\hbox to h_dimen
  {\special{picture name}}}
```

All these packages detail the use of their version of the `\special` command and how each includes graphics into LATEX documents.

Printing the document. Once the PS file has been created, the final step is to process the file through the right printer driver. There are two types of printers available for printing PS documents. For small documents (1 – 20 pages in length), the nearest Apple LaserWriter is used; for large documents (20+ pages in length), one of three Imagen laser printers is used. The Imagens are located in strategic places throughout the Laboratory — normally, far from where you are located.

Conclusions and Recommendations

Recently, graphics created by the high energy physics community, such as PAW and TOPDRAWER, have been placed into LaTeX documents. Experimenting through the years and learning new methods of graphics inclusion, several procedural quick references for Laboratory personnel were developed to show how to include graphics in documents.

A recommendation from one Usenet user on standardization:

> "Unfortunately there is no standard yet for use of `\special` — every DVI processor uses its own scheme. DVI processors should be able to deal with both plain vanilla encapsulated PostScript files (EPS), as well as the 'augmented' encapsulated PostScriptfiles (EPSF) used by some applications, where a low-resolution preview bitmap image in MetaFile or TIFF format is packaged with a PostScriptfile. Most are not able to do this."

By no means am I an expert, but I look forward to continuing to learn more on the subject of graphics inclusion. In this paper I have attempted to share the information I have obtained in researching graphics inclusion in LaTeX from using the `picture` environment to using EPS graphics, briefly discussing how graphics enter user documents, the graphics packages available to accomplish graphics inclusion, and the various uses of the `\special` command for the SSCL printers.

Acknowledgements

I would like to thank Professors Stanly Steinberg and Roger Entringer, and Ms. Moira Robertson with the Department of Mathematics and Statistics at the University of New Mexico for providing me with the opportunity to prove that learning LaTeX was not impossible and for their encouragement in my LaTeX endeavors. Michael Wester and Tom Stickels have also lent me their fine editing skills in putting together this paper. Finally, I would like to thank Ms. Brenda Ramsey of the SSCL who saw in me a hidden talent and provided me with the opportunity to advance my technical expertise. She has been supportive in directing my path toward achieving my long-term goals.

References

[1] Arbortext Inc. *DVILASER/PS User Manual, Version 6.3 for Unix Workstations*, 1985–90.

[2] Blue Sky Research. "Have You Met Your Mac?" *TUGboat* 11(2), advertisement (back cover), 1990.

[3] Blue Sky Research. *Textures Users Guide.* Pages 80–87 and pages 104–105, 1988.

[4] Cisco Systems, Inc. *NetCentral Station Installation and User Guide*, 1990.

[5] Deneba Software. *Canvas Update Manual*, 1988.

[6] Freedman, A. *The Computer Glossary, 4th Edition.* The Computer Language Company Inc., 1989.

[7] Myers, E., and F. Paige. "TEXsis: TEX Macros for Physicists." University of Texas–Austin and Brookhaven National Laboratory, 1990.

[8] Northlake Software. *The T2 Manual: T2/jet, T2/ln03, T2/script*, 1989.

[9] Schwer, L. "Including Macintosh Graphics in LaTeX Documents." *TUGboat* 11(2), pages 194–200, 1990.

[10] Universities Research Association. *To the Heart of the Matter–The Superconducting Super Collider.* Washington, D.C.

[11] Warren, J. "Background Information on the SSC Project." SSCL-501, Superconducting Super Collider Laboratory, Dallas, Texas, 1991.

[12] Weinstein, M. "Everything You Wanted to Know About **PHYZZX** But Didn't Know to Ask." SLAC-TN-84-7, Stanford Linear Accelerator Center, Stanford University, 1984.

Preparing Halftones for Use in TEX

Robert L. Harris
Micro Programs Inc.
251 Jackson Avenue
Syosset, NY 11791-4117
Phone: (516) 921-1351

Abstract

At the 1987 TUG meeting in Montreal, I discussed the issue of incorporating halftones into TEX documents as part of the process of publishing kennel club yearbooks. At that time, the modestly-priced scanners and software did not do an acceptable job, especially when the original was a color photograph.

In the intervening five years, advances have been made in both scanner technology and image processing software.

This is an update, showing what can be expected today when incorporating photographs into TEX documents using the PC platform and low (300 dpi) resolution PostScript printers. Techniques I have used are presented.

Ancient History

Five years ago, I described the production of kennel club yearbooks using TEX. There are two phases to the production of these annuals. The first is the preparation of the dogs' pedigrees. I showed that TEX, combined with a database program, could automate this step, remove the drudgery and produce very attractive, accurate pedigrees.

The second phase was the addition of the dog's photograph to its pedigree. For several reasons, most production errors occurred in this phase. I briefly looked at the technology available at that time to see if the yearbook editor could do this electronically as well as make the camera-ready copy of the pedigree. Without known exception, these annuals are printed in black and white, but the owner-submitted photographs are color prints. Five years ago, modestly-priced software and scanners were not up to the task of producing acceptable images from color prints.

In the past five years, we have witnessed remarkable changes in hardware and software. Whereas an eight megahertz '286 computer was the norm for personal computing then, a twenty-five or thirty-three megahertz '386 or '486 machine is the norm today. The scanner I used for the original work had 400 dpi (dots per inch) resolution and no true grey scale capability — greater for OCR (Optical Character Reader), but poor for processing halftones. When it was used to scan color photographs, on a scale of one to ten, I would have to give its performance a negative value.

Modern History

Although my involvement with kennel club yearbooks has ceased, I have maintained an interest in producing high quality halftone images for use in documents typeset with TEX. I presently produce newsletters for two organizations and numerous publications for my church. The publications frequently are illustrated with halftones. Hence, I thought I would share with you an update to my experiences in this area. As before, I am describing an environment attainable with a modest investment. There certainly is equipment and software that will do far more than I will illustrate — but not within every man's budget.

Hardware. There has been a proliferation of scanners: flatbed scanners, hand-held scanners, autofeed scanners. Most of them have an upper resolution of 300 to 400 spi (spots per inch), although some have double or triple that number. As we shall see later, as in Dalmatians, more is not necessarily better. Virtually all the new scanners are grey scale scanners. The number of grey levels vary. Sixteen or 256 are the most frequently encountered values. Color scanners are being marketed at retail prices that overlap those of grey scale scanners. The color scanners available today in the modest price range use CCD (Charge Coupled Device) technology. This is not necessarily the best technology for color scanning, but is the most affordable.

It does not take long for one to realize that with all this information being generated by the

scanner, it is going to take a lot of memory to process it. Fortunately, today's personal computers have abundant memory and high processing speed. Most people who are going to be processing digital images on a PC-compatible computer will probably be using a '386 or '486 computer with a clock speed of 25 MHz or higher and equipped with two to four megabytes of RAM. With some images requiring over a megabyte of storage, a large hard disk is mandatory if any quantity of images are going to be stored for a period of time.

Since one will want at the very least to preview a scanned image and mostly likely will want to edit — crop, size, retouch — the image, a high resolution display will be part of the system. When a scanned image is displayed on a monitor with 1024 x 768 pixels, one sees a high quality image that is truly representative of what the final image will look like. In some cases, the video representation is better than the hard copy will be!

All of this hardware should cost under $4000 and maybe as little as $3000. The computer with a 130 megabyte hard disk and high resolution display is available at street prices at, or close to, $2000. I prefer a flatbed scanner which is more expensive than a hand-held one. Even so, a high quality one is available for less than $2000.

The astute reader is going to notice that I have not discussed printers. There have been a few printer enhancement products introduced in the past year or so. I am avoiding those because they introduce a device dependence that is an anathema to TEX users. While one could argue that PostScript makes one device-dependent, it is sufficiently widely accepted as to constitute a standard. Therefore, I am restricting myself to output for PostScript devices.

Software. The developments in image-processing software have caught up with the hardware. The software I had five years ago was very limited in its capabilities. The year after I gave my paper, Astral Development Corporation released Picture Publisher, the first greyscale image editing software for the desktop computer. Two years later, they released an upgraded version with color capacity. As the first of its kind — and an excellent product — it became the benchmark for all the image editing software that has followed. Last year, Micrographix acquired Astral Development Corporation. This spring, Micrographix released version 3.0 of Picture Publisher. It requires Microsoft Windows.

Most of the other image processing programs for the PC also required MicroSoft Windows. Here is a quick rundown of the ones of which I have knowledge.

- ZSoft Corporation, one of the early developers of paint-type programs introduced Publisher's Paintbrush last year. They also have a less-expensive program, PhotoFinish.
- Image-In Incorporated has two programs they describe as a "darkroom on a desk": Image-In-Color and Image-In-Color Professional.
- Aldus, through the acquisition of Silicon Beach Software, now has PhotoStyler for Windows.
- Computer Associates has been showing a pre-release beta version of CA-Cricket Image.
- Not all the programs require Windows, however. Mathematica Incorporated has a program called Tempra.
- PixoFoto, like Tempra, does not require Windows. It has its own Windows-like graphical user interface. It does, however, require a special graphics (TIGA) board. This product is from PixoArts Corp.

Once one has chosen and acquired a program, one tends to stay with it unless it is unsatisfactory for the purpose, or fails to stay with the state of the art. Although I own a couple of image processing programs, I find myself using Picture Publisher for all my work.

From this quick survey, tools that are affordable are certainly available. Now the question is how well do they work?

If one is after the highest quality halftones, the best original is still a black and white photograph. The absolute highest quality is going to be achieved by preparing a plate using traditional methods. Given that it is getting increasingly difficult to obtain black and white prints, one may have no choice but to use color photographs. I believe — although I do not have any proof to back up this statement — that a color transparency will reproduce better than a color print.

Halftones prepared from black and white or color prints using digital techniques can be acceptable, depending upon the purpose. I have been using scanned photographs in two newsletters for the past year. It has taken some experimenting to determine the best way to handle different situations. Lets start at the beginning: scanning the image and obtaining the initial digital image. I have a scanner that is capable of 600 spi. This resolution is going to produce very big files. Table 1 shows the memory requirements for an 8 by 10 photograph at a number of scanning resolutions. Fortunately, we do not need all those spots. The resolution we need will be determined by the printer resolution,

Scanner Resolution spi	Greyscale	Full Color
50	0.196 M	0.589 M
75	0.441 M	1.323 M
100	0.784 M	2.353 M
150	1.765 M	5.295 M
200	3.139 M	9.416 M
300	7.061 M	21.182 M
400	12.551 M	37.652 M
500	19.614 M	27.652 M
600	28.242 M	

Table 1. Image Memory Requirements

which we will express as dpi. Table 2 gives the scanning resolutions Micrographix recommends for each printer resolution. The scanning resolution is approximately 1.5 times the screen ruling.

Bob's rule No. 1:
Do not use a higher scanning resolution than you will need for your final output device.

If you are going to use your desktop laser printer, don't waste time and memory scanning at resolutions greater than 80 spi. If you will ultimately be sending your files to a service bureau using a 1270 dpi imagesetter, then use a scanner resolution of 192 spi.

At this point, we also have to decide, since we have a color scanner, whether to scan the photograph as a color image or as a greyscale image. Remember, we will be printing our final document in black and white. Color images will be three times bigger than their greyscale counterparts. While it is fun to edit and manipulate a color image, it does not affect the quality of the output if we do the conversion from color to greyscale when we scan the photograph. Truthfully, it will be it easier to apply proper contrast and/or brightness corrections that may be necessary during image editing if we are working with the greyscale image.

Printer Resolution (dpi)	Screen Rulings	Scanner Resolution (spi)	Grey Levels Simulated
300	53	80	64
400	62	93	83
600	84	126	102
1000	101	152	196
1270	128	192	196
2540	150	225	256+

Table 2. Suggested Scanner Resolutions

Bob's rule No. 2:
Scan your original as a greyscale even if it is in color if you will be printing the final copy in black and white.

What about sizing? It is unlikely that the photograph is going to be reproduced full size in the final document. I prefer to adjust size when I am scanning. It does not affect the image quality and it makes the file size smaller (assuming that I am reducing the size of the picture).

Which brings up an interesting point. If you tried to enlarge an image electronically with earlier scanner software, you were apt to wind up with weird pictures — candidates for the Museum of Modern Art. I have successfully enlarged a photograph by a factor of two when scanning it without any perceptible digitizing in the image.

Akin to sizing is cropping. Perhaps there is unwanted or unnecessary detail in the photograph. Crop it out when you scan the photograph. If you do not get it cropped enough, you can always refine the cropping on the editing desk.

An advantage to doing the sizing and cropping at this time is that the software will do all the adjustments to the scanner resolution necessary to maintain the target resolution. So, set the resolution according to the table and the scanner driver will take care of the rest.

Bob's rule No. 3:
Size and/or crop the photograph when you scan it.

Once you have scanned the image, you can touch it up, adjust the contrast and brightness, do the final cropping and sizing, and save it as an encapsulated PostScript file. I recommend that you make some trial runs on the output. Certain colors will blend together when changed to greyscale. With a good program like Picture Publisher, you can select areas of the image and change the grey value. This is a good way to bring out detail that would otherwise be lost. Manipulating the image

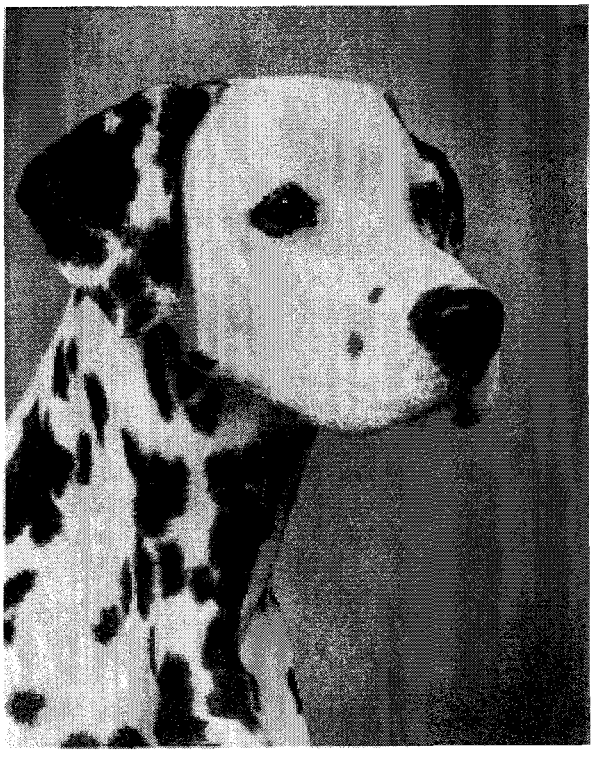


Fig 1. Fashion

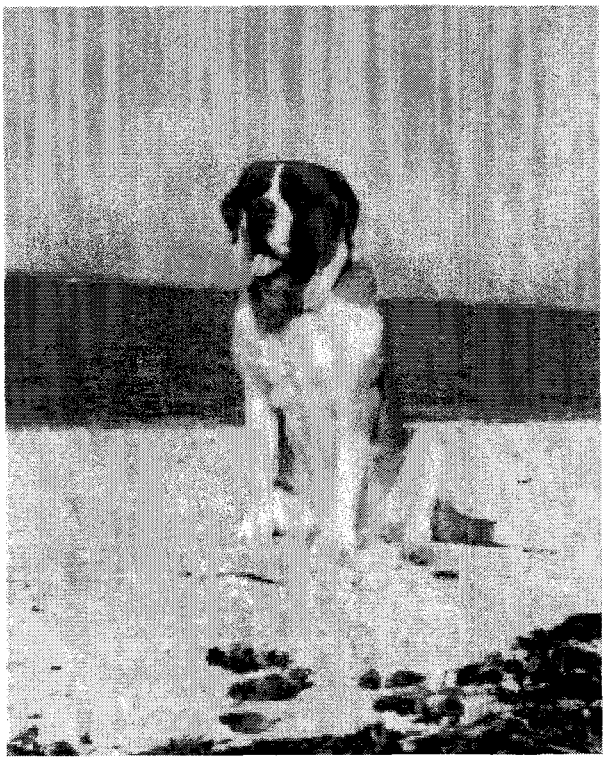


Fig 2. Sneakers

this way can be time-consuming and you probably will not resort to it unless necessary.

Now it is time to view the progress. Let's start with the only photograph from five years ago that gave acceptable results — and borderline at that. It was a black and white photograph of a Dalmatian. Using one of today's scanners and Picture Publisher, she looks a lot better. This illustration was reduced from an 8 by 10 original.

The next example was taken at the beach. The foreground is sand and a little seaweed. The background is water and sky. Sneakers is brown and white with black shadings. The image was cropped from a 4-by 5 color photograph. Five years ago, she would have disappeared into the sand.

The third example is a photograph of a German Shepherd Dog. She is tan and black. She was posed on grass with some evergreen trees in the background. There was a lot of light on her chest — the sun was probably low in the sky. The original was also a color photograph. In this case, the image was enlarged slightly from 2.25 inches to 3 inches.

Lest you think I am prejudiced, the fourth illustration includes some humans. Like the first example, it was prepared from an 8 by 10 black and white photograph.

All four originals were scanned at 80 spi (see Rule 1). The area of the original to be used was masked and the desired image width was set to 3 inches before scanning (see Rule 3). Therefore, the scanner driver took care of adjusting the scanner

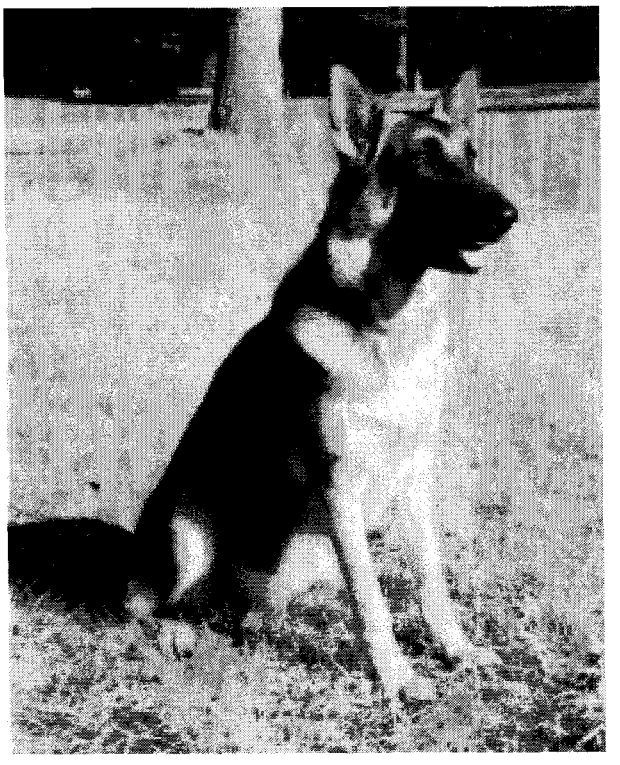


Fig 3. Illsa

Fig 4. Fashion with Judge and Handler

resolution. The two color photographs were scanned as greyscale images (see Rule 2). All four were printed "as scanned". No contrast or brightness adjustments, retouching, filtering or masking was done.

All four images were printed on a 300 dpi printer. Overall, it reminds the viewer of the quality of photographs in the typical daily newspaper of a decade or two ago. As I was preparing this presentation, a number of printer manufacturers were announcing or shipping PostScript printers with increased resolution at prices competitive with today's 300 dpi printers (QMS, for example, now supports 600 dpi on a number of their entry-level printers). The quality of these images would be greater on such a printer.

Depending upon the nature of the publication, one can prepare acceptable halftone images for inclusion in TEX documents today with a modest investment in equipment and software. Perhaps not up to the standards set by *The National Geographic*, but useful for many purposes. The best quality halftones will still be obtained with traditional technology.

Creating Shaded Rectangles with PostScript

David Salomon
California State University, Northridge
Computer Science Department
Northridge, CA 91330 USA
Phone (818) 885-4954; FAX (818) 885-2140
Internet: vacsc0rf@vax.csun.edu

Abstract

One of the most common graphics used in documents is text with a shaded background. This is hard to do with TEX but easy with PostScript. Simple PostScript code is presented here to create shaded rectangles, and a macro is developed to combine such a rectangle with text.

Introduction

It is well known that TEX lacks facilities for graphics. Certain drawings, such as logos, may be developed in METAFONT, but for any non-trivial graphics this is usually too time consuming. A ruled box, like this, is the closest TEX can get to anything resembling a drawing. Slanted lines can be handled by typesetting a dot and moving it in small steps. Hendrickson (1985), Cameron (1985) and Salomon (1989) use this idea to develop simple macros for slanted lines. The same principle can be used to typeset curves, as done by `Bezier.sty` in LATEX, and by the PICTEX macro package (Wichura, 1986). For more complex graphics, the `\special` command can be used to include graphics in the final document. Rahtz (1989) is an excellent survey of the different methods used to combine TEX and graphics.

In my work I have often felt the need for enclosing text in ruled boxes, either rectangular or with rounded corners, with backgrounds of shaded gray, and sometimes with a thick stroke around them. Glendown (1989) is an (unsatisfactory) attempt to draw similar boxes using the arcs provided in the LATEX circle fonts. As a heavy Macintosh user, it seemed to me that my best choice was to use PostScript to achieve such effects. PostScript printers are widely available, and PostScript programs are supported, via `\special`, by more and more TEX implementations. The program used here has the additional advantage of being small and fast.

The `\shade` Macro

The result of my efforts is the macro `\shade` below. It creates a rectangular box with either sharp or rounded corners around text. The text can be placed, by the user, in either an `\hbox` or a `\vbox`, or it can be written explicitly as the macro argument (see third example below). The rectangle is filled with the desired shade of gray, and is optionally surrounded by a stroke of any desired thickness. It is also possible to make the shaded area larger than the text by any amount. The macro has five parameters:

- A decimal number specifying the proportion of white in the shaded background.

This is a .97 background.

- A dimension specifying the

extra size of shaded area (12pt on each side).

- A count specifying the width of the stroke in points. A zero or empty argument creates no stroke.
- A count—the radius of the rounded corners, in points. A zero or empty argument produces square corners.
- The text to be boxed. It is either straight text (if it fits on one line) or is enclosed, by the user, in an `\hbox` or `\vbox`.

The arguments are separated by commas, and the last one is delimited by '\\'.

Here is the listing of `\shade`.

```
\newdimen\tmp \newdimen\tmpw
\newdimen\tmpk \newdimen\tmph
\newcount\heit \newcount\widf
\newcount\strk \newcount\radius
\def\shade#1,#2,#3,#4,#5\\{%
\setbox1=\hbox{#5}%
\def\inner{#3}\ifx\inner\empty \strk=0
 \else\strk=#3\fi
\def\inner{#4}\ifx\inner\empty \radius=0
\else \radius=#4\fi
```

Creation and Incorporation of PostScript Graphics with TEX-formatted Labels into TEX Documents

Neil A. Weiss
Department of Mathematics
Arizona State University
Tempe, AZ 85287
Phone: 602-965-3951; FAX: 602-965-8119

Abstract

In this paper, we will discuss the incorporation of PostScript-created graphics with TEX-formatted labels into TEX documents using the *Mathematical Graphics System*, or *MG* for short. We will provide a brief overview of the creation of such graphics and explain in detail the TEX macros employed to accomplish the inclusion of the graphics and the TEX labels into the output of a TEX document. Additionally, we will provide illustration of how to customize figure insertions for particular applications.

What is the Mathematical Graphics System?

The *Mathematical Graphics System, MG,* is a menu-based program designed for the creation and display of both two- and three-dimensional PostScript graphics on DOS based computers. Although it is not the purpose of this paper to discuss the graphics capabilities of *MG*, we should point out that they are considerable.

Two-dimensional graphs can be specified as Cartesian, polar, or parametric curves, straight lines, and vectors. Three-dimensional graphs can involve Cartesian or parametric surfaces, straight lines, and parametric curves. In addition, surfaces can be ruled by arbitrary families of curves and can be shaded.

MG also has *free-drawing* capabilities where the user can specify points of interpolation for filled or unfilled polygons and for filled or unfilled splines. The scatter-plot feature of *MG* permits the plotting of points read from a data file or for drawing curves or polygons (filled or unfilled) through the points. It also allows for the shading of regions in two-dimensions. For a detailed explanation of the graphics capabilities of *MG*, we refer the reader to Israel and Adams (*Mathematical Graphics System User's Manual*).

The output of *MG* can be obtained in two forms: (1) as a pair of files to be incorporated into the source code of a TEX document or (2) as an encapsulated PostScript file that can either be sent directly to a PostScript output device (printer or phototypesetter) or included in a larger document that will be printed on such a device. In this paper, we will concentrate on the first form of output.

The two files referred to in (1) consist of an `lbl` (label) file and a `ps` (PostScript) file. The `lbl` file contains, among other things, information regarding the TEX-formatted labels and their placement. The `ps` file is a PostScript file delineating the graphic. This latter file is incorporated into the TEX output by employing the `\special` command, the exact syntax of which depends on the `dvi` to PostScript driver being used.

To obtain the `lbl` and `ps` files for a graphic constructed by *MG*, the user first selects "Make PostScript file" in one of the several menus where that option occurs. The user is then asked to choose a file name that will serve as the name for both the file with the `.lbl` extension and the file with the `.ps` extension. Once that is accomplished, the user should specify "importing into TEX" for the type of PostScript file to be saved. (The other option, "printing directly," yields an encapsulated PostScript file with the labels being set by PostScript.) We will discuss both the `lbl` and `ps` files created by *MG* in greater detail shortly.

We should note that the user can also elect to save the `grf` file corresponding to a graphic constructed by *MG*. That file records the menu selections and other information required by *MG* to reproduce the graphic at a later time for editing or electronic viewing purposes.

The LBL and PS Files

Let us now discuss, in detail, the `lbl` and `ps` files obtained when an *MG* graphic is saved using the "importing into TEX" option. The `lbl` file is an ASCII file that contains the positions, justification codes, and text for each of the labels. These labels, which were specified in *MG*, are assumed to be in TEX form; e.g., labels containing mathematical expressions should be enclosed in dollar signs. The `lbl` file also includes information about the dimensions of the PostScript figure as well as a few other pieces of data.

To illustrate, we will examine the `lbl` file of the graphic displayed in Figure 1, the graph of the polar equation $r = \sin 8\theta$:

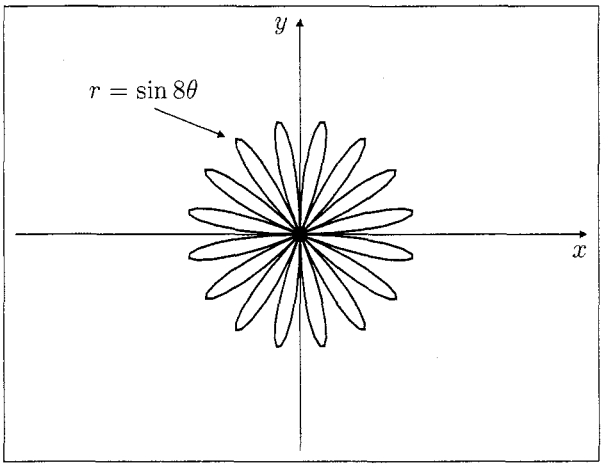


Figure 1: Graph of a polar curve

Note that there are three labels in Figure 1, all of which are set by TEX: the labels for the x and y axes and the label describing the equation of the curve, $r = \sin 8\theta$.

A listing of the `lbl` file that supplies the requisite information to TEX for the proper location and coding of the labels is depicted in Table 1 at the top of the next column. The annotations in the right column of the table are included for explanatory purposes and are not part of the `lbl` file.

In understanding the position coordinates for the labels, it is important to note that the dimensions are in points with reference to an origin at the upper left corner of the figure and that the x-dimension increases to the right whereas the y-dimension increases downward. Since, as we mentioned earlier, the `lbl` file is an ASCII file, it can be edited using a text editor. Thus, to move the equation label, $r = \sin 8\theta$, up five points, we need only change its y-coordinate from 27 to 22.

We now see that the `lbl` file provides the information necessary for TEX to set the labels in Figure 1. The remaining portion of Figure 1 — the axes, graph, and leader — are described in the `ps` file. TEX plays no role in processing the information in the `ps` file except to pass the instructions in the `\special` command to the `dvi` file.

Table 1: Label file with annotation for Figure 1

`101040`	Version of *MG*
`217`	Width of figure, in points
`163`	Height of figure, in points
`104`	x-coord of the y-axis label
`0`	y-coord of the y-axis label
`2`	Right just'n of y-axis label
`2`	Top just'n of y-axis label
`$y$`	y-axis label
`217`	x-coord of x-axis label
`86`	y-coord of x-axis label
`2`	Right just'n of x-axis label
`2`	Top just'n of x-axis label
`$x$`	x-axis label
`49`	x-coord of eqn label
`27`	y-coord of eqn label
`1`	Horiz centering of eqn label
`1`	Vert centering of eqn label
`$r=\sin 8\theta $`	Equation label
`-1000`	End of label file

How MG Incorporates the Labels and Graphics into TEX

In order to incorporate the TEX-formatted labels, specified in the `lbl` file, and the PostScript graphic, specified in the `ps` file, the distribution of *MG* includes a collection of TEX macros in the file `fig.tex`. The macro that provides the commands for the importation of the labels and the calling of the PostScript graphic is called `\figinsert`. For reference purposes, we have presented a listing of that macro in the appendix to this paper.

The argument to `\figinsert` is the common name of the `grf`, `lbl`, and `ps` files. In the case of Figure 1, it is `polar`. The `\figinsert` macro first prepares the input stream, `labelfile`, to read from the specified label file (in Figure 1, `polar.lbl`). Then it reads the first three lines of the label file, which contain the version of *MG* and the width and height of the PostScript figure, to `\mgversion`, `\pswidth`, and `\psheight`. At this time, the name of the file is also displayed on the terminal by use of the `\message` command.

If boxfigures is true, then the figure will be surrounded by a rectangle, as is done in Figure 1. The line width and border width can be changed from the default in *MG* (1 pt and 2 pt, respectively) by altering the values of the dimension registers, \boxrulewidth and \boxborderwidth. In Figure 1, \boxrulewidth=.4pt and \boxborderwidth=4pt.

Next, \figinsert constructs a \vbox whose height is \psheight, the height of the PostScript figure, and uses the \special command to call the PostScript graphic (in Figure 1, the graphic is in the file polar.ps). The actual description of the graphic will be imported into the PostScript file for the main document by the driver. The default syntax for the \special command can be changed to accommodate any dvi to PostScript driver.

The control sequence, \setlabelsize, can be used to alter the default type size for the labels. For example, to change to nine point, we use the definition \def\setlabelsize{\ninepoint}. We should mention that fig.tex includes the \ninepoint and \eightpoint macros provided in Appendix E of *The TEXbook* (pages 414 and 415).

The next portion of the \figinsert macros continues reading the label file using a loop. It begins by reading the fourth line of the lbl file to \xcoord. That line will either be the x-coordinate of the first label or the code number, -1000, for the end of the file. In the latter case, the reading terminates and the final positioning, boxing (if any), and closing of the input stream occurs. Otherwise, the y-coordinate and justification codes are read to \ycoord, \justx, and \justy, and the label itself is read to \label. Then the label is placed in the box \labox which is copied to the appropriate location using the coordinates and justification codes. Iteration now occurs for further labels or the end of the file.

High-level Macros for Figure Format

MG supplies three high-level macros for figure placement and captions. Of course, these macros all call the low-level \figinsert macro.

The first high-level macro for figure placement and captioning is \cfig, whose listing is as follows:

```
\def\cfig#1#2{\par\smallskip
\openin\labelfile=#1.lbl
\ifeof\labelfile\immediate
\write16{Can't find #1.LBL; I quit!}
\end\fi \closein\labelfile
\vbox{
 \centerline{\figinsert{#1}}\smallskip
 \centerline{\figfont#2}}\smallskip}
```

As we can see, the \cfig macro takes two arguments, the figure file name (e.g., polar) and the figure caption. The figure is centered on the \hsize with the caption centered below. Figure 1 uses the \cfig macro.

Note the call to the \figinsert macro. Also note that the caption (#2) is set in the font specified by \figfont. The default for that font in *MG* is boldface but that can be changed as desired.

The second high-level macro supplied by *MG* for figure placement and captions, \rfig, provides for text on the left side of a page and the figure with its caption on the right. This macro takes three arguments: the text, the figure file name, and the figure caption. The width of the text is determined by the width of the figure; namely, it is the \hsize minus the width of the figure, \pswidth, minus 1 pc. As with \cfig, the caption is centered below the figure.

The third high-level placement macro, called \twofigs, allows for two side-by-side figures with captions. Each caption is centered below its figure. The entire display is also centered on the \hsize.

Of course, the user can modify the high-level placement macros supplied by *MG* or define his/her own high-level placement macros. For instance, one might want to increase the skips occurring above and below the figure when applying the \cfig macro, say, by replacing two of the \smallskips by \medskips.

Or suppose, for example, that a book being prepared in TEX uses count registers to keep track of the chapter number and figure number and that the caption of a figure consists of the word "FIGURE" followed by the chapter number, a period, the figure number, and a colon, all set in the font specified by \figurefont, and then by the description of the figure, set in the font specified by \figuretitlefont. Further suppose that figures are insertions using the \midinsert macro of plain.tex. Then the following adaptation of the \cfig macro can be used to accomplish the setting of the figures:

```
\newcount\chapterno \newcount\figureno
\newif\iffigtex

\def\fig#1\par{\iffigtex\relax
 \else \globaldefs=1 \input fig
 \figtextrue
 \def\setlabelsize{\ninepoint}
 \boxrulewidth=.4pt \boxborderwidth=9pt
 \boxfigurestrue \globaldefs=0 \fi
 \goodbreak \midinsert
```

```
\global\advance\figureno by1
\cfig
{fig\number\chapterno-\number\figureno}
{\figurefont \uppercase{Figure}
\number\chapterno.\number\figureno:
\figuretitlefont #1}\endinsert}
```

Note that, at the beginning of the `\fig` macro, TEX checks to see whether the `fig.tex` macros have been input; if they haven't, they are input along with some changes to some of the *MG* parameters.

The following figure, the file name of which is `fig4-1`, was obtained by applying the `\fig` macro. We simply typed

```
\fig A $\Gamma$ section.
```

when `\chapterno=4` and `\figureno=0`.

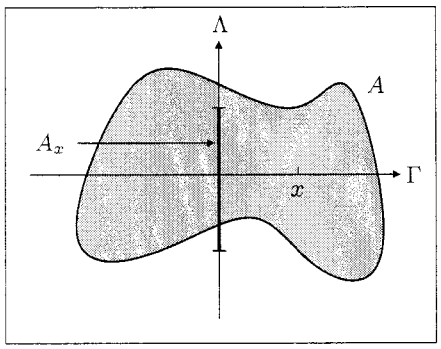


FIGURE 4.1: A Γ section.

Bibliography

Israel, R. B. and R. A. Adams. *MG, Mathematical Graphics System User's Manual.* MG Software, 4223 W. 9th Avenue, Vancouver, B.C., Canada V6R 2C6, 1990.

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

Appendix

The following is a listing of the `\figinsert` macro. Also included are the requisite definitions for that macro. As can be seen, the `\figinsert` macro utilizes both the `lbl` (label) and `ps` (graphic) files corresponding to the graph constructed by *MG*.

```
\newcount\vpos
\newread\labelfile
\newif\ifdoit
\newbox\labox

\def\figinsert#1{\par  %#1=filename
\openin\labelfile=#1.LBL
\global\read\labelfile to\mgversion\message{#1}
\global\read\labelfile to\pswidth
\global\read\labelfile to\psheight
\ifboxfigures\boxit{\fi\vbox to\psheight pt{\vfill
\special{ps: plotfile #1.PS}% Version for DVILASER/PS!
\vskip-\psheight pt \setlabelsize
\hbox to\pswidth pt{\hss}%
\parindent=0pt\offinterlineskip
\vpos=0
\loop\global\read\labelfile to\xcoord
\ifnum \xcoord < -999 \doitfalse\else\doittrue\fi
\ifdoit \global\read\labelfile to\ycoord
\global\read\labelfile to\justx
\global\read\labelfile to\justy
\global\read\labelfile to\label
\global\setbox\labox=\hbox{\label\hskip-0.3em}%
\advance\vpos by-\ycoord
\vskip-\vpos pt \vpos=\ycoord
\hbox to\pswidth pt{\hskip\xcoord pt
\hbox to 0pt{\ifnum\justx>0\hss\fi
\vbox to0pt{
\ifnum\justy<2\vss\fi
\copy\labox\kern0pt
\ifnum\justy>0\vss\fi}\ifnum\justx<2\hss\fi}\hss}
\repeat
\advance\vpos by-\psheight
\vskip-\vpos pt}\ifboxfigures}\fi\closein\labelfile}
```

How to Combine Multiple Languages, PostScript and LaTeX

Timo Knuutila
Dept. of Computer Science
University of Turku
Lemminkäisenkatu 14 A
SF-20520 Turku
Finland

Abstract

A solution on how to handle multiple languages — even the accented ones — with correct hyphenation in standard LaTeX (with the Mittelbach–Schöpf font selection scheme) is presented. Moreover, the solution proposed makes it possible to easily switch between different languages and font families within the same document.

Introduction

Many of us are eagerly waiting for a new LaTeX with support for multi-lingual styles and international characters. There have already been some signs of the shape of things to come: the latest updates for LaTeX 2.09 have made it easier to integrate standard LaTeX with the babel style option of Johannes Braams (1991), and the new font selection scheme (NFSS) (Mittelbach and Schöpf, 1989) is accompanied with styles (currently in beta-test) to use the pre-release of the EC fonts defined in the Cork meeting (Ferguson, 1990) (the DC font family). However, while waiting for the official releases, the casual TeXnician has to manage somehow — usually with his own solutions.

This article concentrates mainly on the *interface* between LaTeX, international characters and languages using these. The language-specific adjustments used at higher levels (e.g., the name of "chapter") can be done with the babel styles. We begin the story with a section describing the reasons I undertook this effort. The current solution to the problems encountered is then presented in more detail.

Working with an Accented Language

In theory. When typing a TeX file, the typist thinks of a *glyph* (shape of a character or a symbol) and hits an appropriate key or key combination. The glyph becomes a small integer number, say i, in the file produced. For example, when working on a PC-compatible computer, the glyph-integer mapping is often defined by the IBM code page 850 (IBM coding hereafter). TeX reads the file and possibly converts i to another code j, which is then used with the precompiled hyphenation patterns. The resulting dvi file then contains integers j serving as pointers to glyphs in a font table. This glyph is finally made visible by the dvi driver that produces the raster image associated with the output code j.

In practice. A Scandinavian typist wants to have the glyph **ä** in his/her TeX text. He/she is forced to continuously type the sequence \"a instead of a single key, since the integer number produced by the key **ä** is not found from TeX's font tables, and neither is the character itself. The glyph is then constructed from two subparts: the accent ¨ and the letter **a**. One could perhaps stand this inconvinience, but accents cause an intolerable difficulty: TeX will not automatically hyphenate words containing accents — we have to write explicit discretionary breaks for them.

Modified versions of TeX do exist, like INRSTeX and MLTeX, which have the ability of hyphenating accented words, but they just are not TeX. In order to make TeX hyphenate properly words containing accented letters we have to create font tables for their representative numbers. However, the mapping from codes above 127 to glyphs (and vice versa) is *not* uniquely defined: there exist (to mention a few) 8 different ISO standards (Latin-1,...,Latin-8), numerous code pages from computer manufacturers, and the extended TeX font encoding scheme (Ferguson, 1990). The last scheme will most probably be accepted as a future standard, but many users of TeX feel reluctant to adopt it because of its incompatibility with the current TeX encoding.[1]

As the EC fonts have not yet been officially released, we describe a slight modification, which uses the Latin-1 coding for the 'upper half' of a 256-letter alphabet, but is downward compatible with current TeX documents. This encoding is referred as the

[1] The Greek letters do not reside in text fonts.

Extended Computer Modern (XCM).[2] These XCM fonts are totally based on the *virtual fonts* (Knuth, 1990) and the standard CM family — no extra METAFONT sources are used. The use of virtual fonts is further justified by the sheer size of a set of new raster fonts for a 256-letter alphabet and all magnifications (or true sizes).

PostScript. The distribution of the dvips driver is accompanied with a program, afm2tfm, which is able to translate the Adobe font metric files (afm) to TEX font metric (tfm) and virtual font (vf) files. Given an afm file as input, afm2tfm creates a font metric file for the raw PostScript characters (no character remapping) and a virtual font property list (vpl) file which defines the standard 7-bit character set using TEX's internal coding. The characters, whose codes are above 127, are mapped to their corresponding Adobe positions. Needless to say, this coding is different from both the Latin-1 coding and from the IBM coding.

We thus have three different codings for a single letter: one for input and two for the output. Because of two output codings, the hyphenation patterns create a further complication due to the need for different \language for Finnish in Computer Modern and for Finnish in PostScript. The only difference in the patterns of these 'languages' are the places where the (codes of) characters **ä** and **ö** are used.

Style files for PostScript. The new font selection scheme makes the definition of the PostScript fonts particularly simple since they are all generated by scaling the same font metric file. There currently exists many LATEX styles illustrating this ease.[3] We have used as a starting point psfonts.sty written by Dick van Soest.

Documents should be written by using only a few different font families — just recall the guidelines for these Proceedings. As an example, we could use Times Roman for plain text, Helvetica for sans serif and Courier for typewriter. Thus, it is not practical to always define all the different PostScript font families, regardless of what fonts are actually used. Therefore, the fonts should be declared *on demand.* The current implementation of NFSS makes the on-demand definition hard or impossible, because all the font declarations have to be made in the preamble of the document. Thus, one should either hack NFSS or explicitly define the fonts to be used in a style file. We have chosen the latter approach, but in order to keep the number of different style files minimal, one file contains all the commands needed to define the different families. The actual definition is made by a single command in the preamble.

Another reason to have one's own style for PostScript is the intermixing of PostScript, multiple languages and standard TEX fonts. Suppose that one writes documents in English and Finnish, both with Computer Modern and New Century Schoolbook (I do it all the time). Since it is faster to process dvi files containing real rather than the virtual fonts, we should use the CM fonts for English, and switch to XCM only when necessary, i.e., when changing the language. This approach also has the extra advantage of compatibility: the dvi files created for documents not using the language switching capabilities remain the same. However, when the XCM encoded PostScript fonts are used, no switching has to be done. Thus, the language switching device must be aware of the font family currently used.

The Evolution of MLATEX

How XCM was chosen. My first attempt to make TEX hyphenate Finnish text (this happened when TEX was younger than three years) was to create the extra glyphs by modifying the METAFONT sources of the Computer Modern font family, replacing some of the Greek characters with the new glyphs, and adjusting the hyphenation patterns according to the coding. Although the solution worked, it was clear that it was a non-portable hack; moreover, the hand-editing and testing of the METAFONT files was very time-consuming — especially for a person who does not know a bit about the programming language used!

Since I was working on a PC-compatible computer, the glyph-integer mapping was defined by the IBM code page 850. These integers (e.g., ^^84 for ä) were then mapped via TEX macros to the codes in the font tables. Knuth actually suggests ligatures (defined in the font property list file) to do the job (*The TEXbook* p. 46). However, this attempt implies key sequences such as a" should be used, which is not what was wanted.

Then came TEX 3.0 with 8-bit input and the facility of virtual fonts (Knuth, 1989). An instant idea was to exploit these facilities and create a virtual font on the top of each TEX font by adding the Scandinavian letters somewhere above the 7-bit barrier. At that time it seemed natural to use the

[2] It is an unfortunate coincidence that the Russian TEX project has adopted the same naming for their Cyrillic fonts (Malyshev et al, 1991).

[3] The definition of the CM fonts LATEX uses would be almost as easy if one used the sauter fonts (available from ftp.cs.umb.edu).

IBM coding for the foreign letters, because no extra transformations were then needed to process the documents. The process itself was straightforward: take a `tfm` file, convert it to a property list file (`pl`) with `tftopl`, hand-edit the result and create the `vpl` file, and finally run `vptovf` which gives the `vf` and `tfm` files needed. The hand-editing could have been done automatically, but I did not have the time (and skill?) to construct a program to do it.

As time passed, I became aware that there is a standard for the 8-bit character codes, namely ISO8859-1 also known as Latin-1. Tor Lillqvist (`tml@tik.vtt.fi`) from the Finnish Technical Research Center had created a set of virtual fonts, Extended Computer Modern, which used this standard. Actually, these `XCM` fonts do not contain all characters of Latin-1. Only the ones that can be constructed from the accents and characters of Computer Modern are included.[4] Moreover, the letters æ, œ and ß are located in their traditional places.[5] Tor had also written an Emacs Lisp macro which extends any property list file of `CM` fonts to a virtual property list file using the aforementioned `XCM` coding. The `XCM` fonts, which contain the standard font family of TeX, are created with this program.

Mapping input to `XCM`. The following problem was to remap the IBM codes to the `XCM` codes, but this was easily solved by declaring the extra characters as active. For example, the following definitions were needed for the letter ä:

```
\catcode`^^84=\active
\let^^84=^^e4
```

Later, I learned to utilize the TeX code page utility (`TCP`) of the emTeX-package (Mattes, 1990), and these kinds of macros were no longer needed.[6]

In 7-bit editor environments (e.g., a UNIX workstation and Gnu Emacs) there were two possibilities. In the first one the text is edited on a personal computer and then sent to the mainframe to perform the TeX-processing (and possibly other duties such as spell-checking). The character codes are mapped as explained above with a style file. Another possibility is to write documents as usual (using accents) on the workstation and redefine TeX's accents according to the `XCM` coding, again in a separate style file. As an example, `\"a` is translated to `^^e4` by the following redefinition of the ¨-accent:

```
\gdef\"##1{%
   \if##1a{^^e4}%
   %similar \if's for other
   %"-accented characters
   \else {\accent"7F ##1}\fi}
```

Actually, there is also a third way to write Finnish documents. The Scandinavians are used to replacing their national characters with letters found on a US keyboard. For example, { stands for ä, | stands for ö, etc. There exists a Scandinavian LaTeX, SLaTeX, designed with this coding in mind. Our system supports this implementation by mapping the SLaTeX characters to the `XCM` coding via a macro file.

Unifying PostScript and `XCM`. The first attempt to unify the PostScript and `XCM` codings was based on macros and character remapping: whenever PostScript fonts were used, each `XCM` code of a Scandinavian letter was defined to be active and mapped to the Adobe coding. Each PostScript font family (e.g., Times Roman) was associated with a LaTeX environment in which this mapping took effect. In 7-bit environments the accents had to be mapped similarly to their correct values.

Analogous to the character mappings, whenever a transition from the usual TeX coding to the Adobe world was made, the `\language` had to be adjusted, too. The resulting system, though working, was apparently quite clumsy and space-wasting.[7]

In order to have only one (`XCM`) set of hyphenation patterns for each language used, the Adobe codes had to be mapped somehow to their Latin-1 codes. The best solution was to modify `afm2tfm` in such a way that it mapped the characters (above 127) in the `vpl` file to their `XCM` codes, not to the Adobe ones. Actually, only a very slight modification to this program was needed. It should be noted that the latest version of `afm2tfm` (v7.0) (distributed with `dvips` v5.485 and up) allows different encodings for PostScript fonts. However, this facility seems to be still under developement and "for wizards only".

An Overview of the MLaTeX System

The MLaTeX system consists of the following parts:

[4] The `DC` fonts contain a full implementation of Latin-1.

[5] The PostScript fonts created by `afm2tfm` locate these letters similarly by default.

[6] The code page facility of emTeX can be used to map character codes to others before they are passed to TeX's mouth.

[7] Fortunately, the Finnish language is very orthogonal with regard to hyphenation: the pattern file takes less than 4 kilobytes in ASCII. The situation is much worse for more complex languages such as German, which needs over 40 kbytes of hyphenation patterns.

- files needed to create a new LaTeX format,
- LaTeX styles for multiple languages and PostScript fonts,
- XCM encoded virtual fonts for Extended Computer Modern and PostScript, and
- miscellaneous utility programs and files.

The main design principle has been *compatibility* with existing styles (babel, SLaTeX) and documents; no changes are needed for old LaTeX documents written in English. In addition, the dvi files for old documents remain identical.[8]

MLaTeX format files. The files used in the construction of the MLaTeX format can be divided in two parts: the font definition files (for NFSS) and the files needed for hyphenation. We currently have two replacements for fontdef.tex: fontdef.xcm and fontdef.xcm-sauter.[9] They both define a corresponding XCM font family and shape for each CM font (excluding the symbol fonts, of course).[10] The difference between these files is that fontdef.xcm-sauter uses true point sizes whereas fontdef.xcm uses the traditional scalings. For example, cmcsc20 is used instead of cmcsc10 at 20.74pt.

The hyphenation files contain a replacement for the master hyphenation file of LaTeX, lhyphen.tex, named lhyphen.mlatex. It defines the languages used and loads their hyphenation patterns. The languages are named in the form l@language, e.g., l@english, for compatibility with the babel styles. Currently, there are patterns for English, Finnish, German and Swedish. The pattern files are named hyphen.english, etc.

Files latin-1.tex and latin-1-xcm.tex contain the definitions for the \charcode, \uccode and \lccode of each Latin-1 character. In the latter file, only the characters contained in the XCM fonts are considered. It should be noted that latin-1.tex is of use with *any* implementation using Latin-1 encoded fonts (such as the DC fonts). The master hyphenation file loads either one of these files before the hyphenation patterns. Currently, latin-1-xcm.tex is the default. The essential contents of latin-1.tex are shown below.

```
\begingroup% save counters
% Make a loop from #1 to #2,
% change case at #3.
\def\setrange#1#2#3{%
\newcount\isochar%
\newcount\casechar%
\isochar=#1%
\loop%
  %We are in a group, hence global.
  \global\catcode\isochar=11%
  \casechar=\isochar%
  \ifnum\isochar<"#3% uppercase?
     \advance\casechar by "20%
     \global\lccode\isochar=\casechar%
     \global\uccode\isochar=\isochar%
  \else%
     \advance\casechar by -"20%
     \global\lccode\isochar=\isochar%
     \global\uccode\isochar=\casechar%
  \fi%
  \advance\isochar by 1%
\ifnum\isochar<#2\repeat}%
%
\setrange{"80}{"BD}{"A0}%
\setrange{"C0}{"FF}{"E0}%
\endgroup%
```

Style files. The styles are named multi (for *multiple* languages) and ps-nfss (for PostScript fonts with NFSS). Loading the multi style defines a macro \set<language> for each language defined in lhyphen.mlatex[11] (for example, \setswedish). The default language is English.

Style ps-nfss provides two commands for each PostScript font to be used — one for the declaration and one for the actual use. Fig. 1 contains a table of the commands currently defined.[12] Each font used in the document must be declared with a \load command in the preamble. However, Courier and Helvetica fonts are always defined, since they are commonly used for \tt and \sf. For changing back to Computer Modern, the command \computermodern is provided.

Both the commands for changing the font family and the language are designed to work properly with grouping i.e., they are in effect only within the group they are actually used. Below is an example on the use of these styles.

```
\documentstyle[multi,ps-nfss]{article}
\loadtimesroman\loadavantgarde
\timesroman
```

[8] It would be technically easier to always use the XCM fonts regardless of the language used, but then we would lose this property.

[9] Different names must be used for systems with a restricted length for file names (e.g., MS-DOS).

[10] The AMS symbol fonts, Euler fraktur fonts and the Cyrillic fonts from the University of Washington are declared, too. The user may comment out these declarations in the event these fonts are not needed.

[11] If the user of this system modifies the master hyphenation file by adding and/or removing languages, the style file should be edited, too.

[12] The lengthy names have been chosen on purpose in order to inhibit wild font abuse.

Font family	Loading	Using
Avantgarde	\loadavantgarde	\avantgarde
Bookman	\loadbookman	\bookman
Courier		\courier
Helvetica		\helvetica
NCS	\loadnewcentury	\newcentury
Palatino	\loadpalatino	\palatino
TimesRoman	\loadtimesroman	\timesroman
ZapfChancery	\loadchancery	\chancery

Figure 1: PostScript fonts supported by the ps-nfss style.

```
\begin{document}
This text is output in Times Roman.
{\avantgarde
This text is output in Avantgarde.
\setfinnish
Here we are, trying to hyphenate
English with Finnish patterns.
But still Avantgarde.
\computermodern
We won't give up trying to hyphenate
English with Finnish patterns.
Extended CM is used.}
English text, output in Times Roman.
\end{document}
```

Font files. The fonts consist of virtual font files and font metric files both for the extension of the standard LaTeX set of CM fonts and for the raw PostScript fonts distributed with the dvips driver. The XCM fonts contain the font families corresponding to the CM families cmr, cmti, cmsl, cmcsc, cmbx, cmbxsl, cmbxti, cmss, cmssbx, cmssi and cmtt with point sizes 5 – 12,14,17,20 and 25. One can always create his/her own new virtual font from an existing tfm or afm file by using the utility programs extend-cm.el and afm2tfm-iso described in the sequel. The tfm and vf files for the non-raw PostScript fonts found in the dvips package should be replaced with the new ones.

Utility files. The following is a brief description of the miscellaneous utility files. Some of them are usable only with a certain style package or with a specific TeX implementation.

mapacc.sty: A style file which redefines TeX's accents and maps them to the XCM character codes.

afm2tfm-xcm.c: A modification of afm2tfm using the XCM instead of the Adobe coding.

extend-cm.el: An Emacs Lisp macro package which can be used to create an extended vpl file from an existing pl file.

850-xcm.txt: emTeX code page for the translation from the IBM code page 850 to the XCM codes.

ibm2iso.tex: A macro file defining a mapping from the IBM code page 850 to the XCM codes.

ml-swedish.tex: A macro file to be used with the SLaTeX package (a replacement for swedish.tex).

ml-babel.hyphen: A multi-aware master hyphenation file for use with the babel styles. A similar ml-babel.switch is also provided.

Technicalities of the Style Files

We explain in more detail the implementation of the most important commands of multi and ps-nfss. The central thing is the co-operation of these styles. Before that, let us describe the stand-alone commands of ps-nfss.

Suppose that the command

```
\declare@psfont#1#2#3#4
```

defines the font family #1 with shape #3 and series #4 by using the font metric file #2.tfm for all standard LaTeX magnifications. Then the macro \declare@std below can be used to declare all the standard shapes and series for a usual PostScript font.

```
% declare@std{family}{normal}{italic}
% {slanted}{smallcaps}{bold}
\def\declare@std#1#2#3#4#5#6{%
  \declare@psfont{#1}{m}{n}{#2}%
  \declare@psfont{#1}{m}{it}{#3}%
  \declare@psfont{#1}{m}{sl}{#4}%
  \declare@psfont{#1}{m}{sc}{#5}%
  \declare@psfont{#1}{b}{n}{#6}%
  \extra@def{#1}{}{}}
```

For example, the command \loadavantgarde is defined as follows:

```
\def\loadavantgarde{%
  %Prevent double declaration.
  \@ifundefined{ava@loaded}{%
    \declare@std{ava}{pagk}{pagko}%
                {pagdo}{pagkc}{pagd}%
    \def\ava@loaded{}}{}}
```

When a PostScript font is to be used, we usually set the \sfdefault to Helvetica, \ttdefault to Courier and bfdefault to b. [13] This is done by macro \set@defs. The command go@PS#1 shown below switches to a given PostScript font family:

```
\def\go@PS#1{%
  \let\in@ps=1%
  \def\default@family{#1}%
  \def\rmdefault{#1}%
  \set@defs%
```

[13] For the other default values, we use the ones defined by NFSS.

```
\fontfamily{#1}\selectfont}
```

The previous macro is then used to implement most of the font change commands, for example:

```
\def\avantgarde{\go@PS{ava}}
```

The two styles are aware of each other by checking the existence of certain macros. The style multi defines a macro in@xcm, which indicates whether we are currently using a language exploiting the XCM fonts or not. Similarly, ps-nfss defines an indicator in@ps which tells if we are currently using a PostScript font family.

These flags are used in the following two situations.

1. Changing a language may cause a switch between CM and XCM, but this kind of action is not needed when a XCM encoded PostScript font is used. Thus, the \set commands check whether the ps-nfss style is loaded and, if so, the value of in@ps tells the current situation.
2. The command \computermodern switches to CM or XCM depending on the current language. However, if the multi style is not loaded, we always change back to CM.

The implementations of the commands \computermodern and \setfinnish are given below. Commands go@cmr, set@cm and set@xcm set the default values likewise to \go@PS.

```
\def\setfinnish{%
   \set@language{\l@finnish}{2}{2}%
   \let\in@xcm=1%
   \@ifundefined{in@ps}{\set@xcm}{%
      \if0\in@ps\set@xcm\fi}}%
\def\setenglish{%
   \set@language{\l@english}{2}{3}%
   \let\in@xcm=0%
   \@ifundefined{in@ps}{\set@cm}{%
      \if0\in@ps\set@cm\fi}}%

\def\computermodern{%
   \let\in@ps=0%
   \@ifundefined{in@xcm}%
      {\go@cmr}%
      {\if0\in@xcm\set@cm%
        \else\set@xcm\fi}}%
```

The actual change of the language is accompanied by the setting of the left and right hyphenmins (see below). It is the author's opinion that the values 2 and 3 are not the best possible for all languages on earth.

```
\def\set@language#1#2#3{%
  \language#1%
  \lefthyphenmin=#2%
  \righthyphenmin=#3}
```

Conclusion

We have presented an extension of LaTeX capable of handling multiple languages and PostScript fonts. The system is compatible in many ways, and even an English-speaking (and writing) LaTeX user might be tempted to install and use it. This is because the files for the XCM fonts do not take much space,[14] and the old documents do still turn into the same dvi files they used to. Yet, changing to international and/or PostScript is just a flip of a switch.

The development of a fast multi-lingual LaTeX (MLaTeX or not) would be much easier if we had the capability to load hyphenation patterns without initex. An even better solution would be to *precompile* the patterns (something like \dumppatterns) and then load and unload the precompiled hyphenation files dynamically. I also wish that we could adopt the sauter fonts as a standard for LaTeX. This would make the documents look better, and the definition and usage of all the CM fonts could be done in an orthogonal manner.

Bibliography

Braams, Johannes. "Babel, A Multilingual Style Option for Use with LaTeX's Standard Document Styles." *TUGboat* 12(2), pages 291–301, 1991.

Ferguson, Michael. "Report on Multilingual Activities." *TUGboat* 11(4), pages 514–516, 1990.

Knuth, Donald E. *The TeXbook*. 15th ed. Reading, Mass.: Addison-Wesley, 1989.

Knuth, Donald E. "The New Versions of TeX and METAFONT." *TUGboat* 10(3), pages 325–328, 1989.

Knuth, Donald E. "Virtual Fonts: More Fun for Grand Wizards." *TUGboat* 11(1), pages 13–23, 1990.

Malyshev, Basil, Alexander Samarin, and Dimitri Vulis. "Russian TeX." *TUGboat* 12(2), pages 212–214, 1991.

Mattes, Eberhard. *emTeX 3.0 User Manual*. 1990.

Mittelbach, Frank, and Rainer Schöpf. "A New Font Selection Scheme for TeX Macro Packages—The Basic Macros." TeX Users Group 10(2), pages 222–238, 1989.

[14] The total amount of space taken is about 330 kbytes.

Just Give Me a Lollipop (It makes my heart go giddy-up)

Victor Eijkhout
Department of Computer Science
University of Tennessee at Knoxville
Knoxville TN 37996-1301
Internet: eijkhout@cs.utk.edu

Abstract

The Lollipop format is a meta-format: it does not define user macros, but it contains the tools with which a style designer can easily implement such user macros. This article will show some of the capabilities of Lollipop and will give the reader a small peek behind the scenes of the implementation.

> TEX is intended to support higher-level languages for composition
>
> *Donald Knuth*

Introduction

One of the reasons that TEX is not widely accepted outside the scientific world is that the effort needed to create new visual designs, or even to make minimal modifications of a given design ("this article is a bit too long, but since we have rather generous margins, why don't we put the title in the margin next to the abstract, instead of over it") is disproportionally large. In Eijkhout and Lenstra (1991) it was argued that one way of solving this problem would be to implement powerful tools that a style designer could use to program macros without ever programming in TEX itself. In effect, the style designer "needs only say what she wishes done" (Perlis) and the meta-format creates the macros that do this. This article describes such a meta-format: Lollipop.[1]

Now, for those who wondered at the title of this article, the first half refers to an epigram by Alan Perlis, to be found on page 365 of *The TEXbook*; the second half derives from a sixties ditty by Millie Small. All other etymologies are erroneous, and severely frowned upon.

The Structure of Lollipop

The Lollipop format tries to provide tools that make programming macros as hard as using them. I will not discuss the use of the resulting macros in detail, but will focus on implementational matters.

Working with Lollipop. In order to process a document in Lollipop there has to be a 'style definition' for that document. This definition, a sequence of Lollipop macro calls, can be in the document itself, it can be `\input`, or it can be contained in a format. The latter option of loading a style definition in Lollipop and dumping the result as a new format is encouraged for two reasons. First of all, it indicates better the separation between the work of the style designer and that of the user. Secondly — especially on old computers (say of the order of a 286) — processing the style definition for a complicated document can easily take one or two minutes.

The basic Lollipop macros. The Lollipop format is partly a macro collection — and some of the more interesting utilities will be discussed below — and partly a tool box for defining macros. The tools are four macros for defining

- headings (`\DefineHeading`): the main characteristics of a heading are that it has a title, and that it should stay attached to the following text;
- lists (`\DefineList`): a list is characterized by the fact that it has items;
- text blocks (`\DefineTextBlock`): a text block is basically just a group, however, it is so general that lists and headings are really special cases of text blocks; and
- page grids (`\DefinePageGrid`): a page grid is (a macro that installs) an output routine.

Each of these macros[2] can have a large number of options.

An example of the use of Lollipop. Although a large number of examples would be necessary to give a representative sample of the possibilities of the Lollipop tools, here is one example to give the

[1] The Lollipop format is available for anonymous ftp from `cs.utk.edu`.

[2] There is in fact a fifth macro `\DefineExternalItem`, closely related to `\DefineList`; it will be treated below.

reader the basic idea. The following macro defines a heading \SubSection.

```
\DefineHeading:SubSection counter:i
    whitebefore:18pt whiteafter:15pt
    Pointsize:14 Style:bold
    block:start SectionCounter literal:,
          SubSectionCounter literal:.
          fillupto:levelindent title
    external:Contents title external:stop
    Stop
```

(The terms 'block', 'external', et cetera, are called 'options'.) This definition specifies that subsections have a counter that counts in lowercase roman numerals, that there should be a certain amount of white space above and below it, and that it should be formatted in 14 point bold as the section counter, a comma, the subsection counter, a full stop, filling these counters up to the \levelindent width (to be explained below), and following this by the title. Also it specifies that the title should go to the contents file.

This macro \DefineHeading must be a pretty complicated object, don't you think? Well, here is the full definition:[3]

```
\@GenericConstruct{Heading}
\def\@DefineHeading{
    \@DefineStopCommand{\relax}
    \csarg\edef{\@name}{\@GEN@OPEN
        \the\@main@options@list
        \@GEN@CLOSE}
    }
```

where the auxiliary macro \csarg is defined as

```
\def\csarg#1#2{\expandafter#1%
    \csname#2\endcsname}
```

Definition of the \Define macros. Since the \Define... macros are so much alike—many options are common to all of them I let all of them be defined automatically by the same macro \@GenericConstruct. This defines \DefineHeading as a macro that will process a list of options (this part contains the common work for all constructs), and then call \@DefineHeading to do the actual definition.

A call \DefineHeading:Section will expand first of all to a call

```
\def\@name{Section}
```

As can be seen in the example above, this macro is then used to define \Section with an \edef. This \edef unpacks the token list \@main@option@list that has been constructed during option processing. Also, the macros \@GEN@OPEN and \@GEN@CLOSE contain lots of conditionals that may or may not cause code to be included in the definition of \Section depending on values of parameters that were set during option processing. This is explained further below.

Options. Clearly, a large responsibility rests on processing the options. For instance, in the example above the option 'counter' has to allocate the appropriate counter, but also set the test \has@countertrue.

Options can be general, such as the 'counter' option (here \xp is \expandafter):

```
\@GenericOption{counter}{\has@counteryes
    \NewCounter:\@name
    \xp\add@mark@item\xp{\@name Counter}
    \CounterRepresentation:\@name=#1
    }
```

or they can be specific, such as the option controlling white space between items in a list:

```
\@ListOption{whitebetween}{....}
```

Generic options are defined as follows:

```
\def\@GenericOption#1{
    \append@to@list
        {@GenericOptions}{\\#1;}
    \csarg\def{Option@#1}##1##2}
```

for instance, for 'counter' a macro \Option@counter is defined. The definition

```
\@GenericConstruct{List}
```

causes the definition of \@ListOption:

```
\def\@GenericConstruct#1{
    \csarg\def{@#1Option}##1%
        {\csarg\def{#1@##1}####1####2}
```

so that the 'whitebetween' option causes the definition of a macro \@List@whitebetween.

Now let's say we are defining a heading, and we find the option 'foo'. We then check whether a macro \Heading@foo is defined. If so, we execute it; if not we check for the existence of a more general macro \Option@foo. This is executed if it exists, and if it doesn't, we check whether \foo is a defined control sequence. If it is, we include the command \foo in the \@main@options@list, so that it will later be part of the definition of the heading we are defining; if it is not, we generate an error message.

The Basic Tools

In this section I will give a short overview of the capabilities of the four basic macros. First the block

[3] Several pieces of code in this article have been simplified. Others however, such as the following, have been left intact to convey to the reader the idea that Lollipop is a sophisticated format.

structure macros used in all of them are explained briefly.

Block structure. Text blocks and lists are obvious candidates for environment macros that do grouping, so that values of `\leftskip`, `\parindent`, and whatever more can stay local. As I've argued in (Eijkhout, 1990) such macros can also handle spacing above and below the environment. Thus, Lollipop has two macros

```
\def\@GEN@OPEN{\outer@start@commands
    \begingroup \inner@start@commands}
\def\@GEN@CLOSE{\inner@end@commands
    \endgroup \outer@end@commands}
```

that induce grouping, and that perform the various actions needed at the boundaries of an environment. This also includes such common actions as handling counters and titles, placing marks, and defining labels for symbolic referencing.

For instance, if the macro currently being defined (if this is `\Section`, the macro `\@name` has that value) has a counter, that should be incremented. Therefore the macro `\@GEN@OPEN` contains a line

```
\ifhas@counter
    \noexpand\StepCounter:\@name
\fi
```

Recall that these macros are called inside an `\edef`, so `\Section` macro contains the line

```
\StepCounter:Section
```

only if the macro has indeed a counter.

In general, a macro `\foo` opening the environment will contain the code generated by `\@gen@open`, while a corresponding command `\foostop` contains the `\@gen@close` code.

Headings: Maybe somewhat surprisingly, a heading can be considered as an environment, namely as one where the heading command contains both the opening and closing commands of the environment. Titles are treated below.

Text blocks: Text blocks are environments that can span several paragraphs. They have explicit open and close commands. Text blocks are, for instance, a way of having a chunk of text be indented and perhaps labeled. As an example, here is the specification of the examples in *TEX by Topic* (Eijkhout, 1992): they are indented, and the word 'Example' is set in italic over them.

```
\DefineTextBlock:example
    breakbefore:500 breakafter:1
    PushListLevel
    noindent begingroup Style:italic
        literal:Example endgroup
    par nobreak Indent:no
    text
    Stop
```

The option '`text`' indicates where the text of the block fits in the specification. Any options appearing after this option will result in code in the macro that closes the environment. For instance, here is a possible way of defining left-aligning display equations:

```
\DefineTextBlock:DisplayEq
    whitebefore:abovedisplayskip
    whiteafter:belowdisplayskip
    whiteleft:levelindent
    literal:$ displaystyle text
    literal:$
    Stop
```

The closing macro will be defined as

```
\def\DisplayEqstop{ ...
    $
    \endgroup ... }
```

where the dollar corresponds to the one after the '`text`' option.

Lists: The main point of interest about lists is the formatting of the item labels. The two main choices are

```
item:left ... item:stop
```

for left-aligning, and

```
item:right ... item:stop
```

for right-aligning labels. The label can for instance contain an '`itemCounter`', or an '`itemSign`', or even both. The item sign and the representation of the item counter are dependent on the level, and can be set by the designer.

An interesting option is '`description`'. If this option is used, all text following `\item` to the end of line will be taken as the label text. The LaTeX style description list can be implemented as

```
item:left Style:bold
  description Spaces:2
  item:stop
```

which is used as

```
\item The label
 and the next line is again normal
```

Abbreviated closing: Both lists and text blocks have an explicit closing command. Since such phenomena are properly nested, the format can very well figure out what to close if I tell it to

close the current block. Therefore, the macro \> closes whatever list or text block is opened last, and \>] closes all lists and text blocks that are currently open.

Page grids: Definition of output routines is much easier in Lollipop than in plain TEX, but still it is the hardest part of working with Lollipop. Hence I will not go into full detail.

The most important option for page grids is 'text'. It indicates that a page will use part of \box255. If this option does not appear, we are defining an output routine that does not use \box255. For such output routines the option \nextpagegrid is crucial: it tells TEX what output routine to take when the current one has output a page.

For instance, if left and right hand pages have a different layout, we could implement them as separate output routines:

```
\DefinePageGrid:leftpage
    nextpagegrid:rightpage
    ...
\DefinePageGrid:rightpage
    nextpagegrid:leftpage
    ...
```

The 'text' option usually appears inside

```
band:start text band:end
```

and it can occur several times there. For instance

```
band:start text
  white:20pt text
  band:stop
```

defines a two column layout with a gutter width of 20 point.

Some of the options for page grid (height and width for instance) have a global significance, but for others it is recorded whether they appear before or after the 'text' options. Depending on this, they become part of the header or the footer of the page.

When the output routine is invoked, Lollipop assembles any header or footer, and computes the remaining space for text. If this is not equal to the size of \box255, \vsize is reset, and \box255 is thrown back to the main vertical list. This mechanism is an easy way to get pages with the same size if the size of the header or footer can vary.

Definition of output routines is in fact so easy in Lollipop that for title pages of chapters it is easier to write a special page grid, than to mess around with a lot of macros. Thus the line

```
\Chapter The second chapter\par
```

may look to the user as calling a macro, whereas in fact it installs a new output routine for the chapter title page. The way the title is handled is explained below.

Titles and References

The perceptive reader may have noticed in the definition of \DefineHeading above that the macro defined is not declared with a parameter. How then are titles handled?

Well, since in Lollipop not only headings, but also lists, text blocks, and page grids can have titles (but need not; every once in a while a heading without a title can be convenient, and output routines with titles are surprisingly useful, as I indicated above), the option 'title' controls whether a construct actually has a title by setting a switch \ifhas@title to true. Definition of the actual heading macro then executes a line

```
\ifhas@title \@Titelize{\@name}\fi
```

where \@Titelize is a macro that takes a macro, and redefines it with an argument.

This redefinition trick can even be performed twice: if the macro has a counter, this should be referenceable. For some reason I decided against the LATEX approach of using \label commands: any command that can be referenced in Lollipop[4] accepts an optional parameter with the label key. For instance

```
\Section[definition:section] Notations
    and Definitions\par
```

gives the key 'definition:section' to a section.

Indentation Levels

If lists of various types are used in a nested fashion, each next level is indented with respect to the previous one by a certain amount. Specifying these amounts can be done quite flexibly in Lollipop, and it is also made easy for the designer to have other indented material line up with these implied left margins (Braahms, Eijkhout and Poppelier, 1989).

On each level, a control sequence \levelindent indicates the amount by which the next level will be indented. Thus, letting \parindent be set equal to \levelindent at the start of a text block, will give nicely aligning indentations no matter at what level the block appears.

[4] Not explained in this article is that the way something is referenced is also easily determined by the user. This makes it possible for instance to refer to chapters by name instead of by number.

The value of `\levelindent` is determined by looking at the level number (say that this is 3), and checking whether a macro `\levelindentiii` exists. If so, the value of this is taken, if not, some default fraction of the value of `\basicindent` is taken. The style designer can set this `\basicindent`, and adjust individual levels by

```
\LevelIndent:3=25pt
```

or similar calls.

Lists are indented to the next level automatically, but in order to provide this functionality for other objects there exists an explicit

```
\PushListLevel
```

command. There is even a `\PopListLevel` command that has various uses. For instance, it can be used to realise 'suspended lists': the effect of

```
\item Some text\par
{\PopListLevel
\noindent Some text.\par}
\item Again an item
```

is that the 'some text' aligns with the text outside the list, instead of with the items in the list.

Popping and pushing list levels is also essential for correct formatting of external files; see below.

Marks

TEX's marks are a means of communication between routines that supply certain information (values of counters, titles), and the output routine. Since there is no way for the output routine to tell the rest of the macros which ones should pass information through marks, in Lollipop *everyone* puts their information (that is, titles and counter values) in marks. The output routine then selects with a simple call, for instance

```
\LastPlaced:SectionTitle
```

the value of `\SectionTitle` in the `\botmark`.

Let's look at the implementation of this. There is a list `\mark@items` that has the names of everything that goes in a mark. For instance, defining a heading `\Section` causes calls

```
\add@mark@item{SectionTitle}
\add@mark@item{SectionCounter}
```

These allocate token lists

```
\mark@toks@SectionTitle
\mark@toks@SectionCounter
```

which are to contain the title and the counter value, and which get their value from a command such as

```
\refresh@mark@item
   {SectionTitle}{The title}
```

whenever `\Section` is called. Everytime a mark item is refreshed, a new mark is placed on the page which contains the values of *all* mark token lists. The output routine then simply picks from this structure whatever information it needs.

External Files

Formats such as LATEX usually supply facilities for a table of contents, and maybe lists of figures and tables, but what if an author needs in addition a list of notations, one of definitions, and one of authors referenced? Lollipop takes the drastic approach, and provides none of these.

But it makes it easy for you to define them yourself.

User interface. An external file is characterized by in an internal name, and a file name extension:

```
\DefineExternalFile:Contents=toc
```

This command does some initialization such as a call to `\newwrite`, and it creates a switch so that the user can specify with

```
\WriteContents:no
```

that the file is not to be overwritten in this run. A global switch

```
\WriteExtern:no
```

prohibits all external writes.

Next, commands such as `\Section` have to specify that they want to write out information. This is done with the option '`external`'. Usually, all that is written out is the title

```
external:contents title external:stop
```

but other information can be written out too.

The hard part about external files is specifying how their information is to be typeset. Telling that a file needs to be loaded is simple:

```
\LoadExternalFile:Contents
```

For every command that writes information to an external file, the style definition needs to contain a call

```
\DefineExternalItem:Section file:Contents
    item:left Style:roman SectionLabel
        item:stop
    title Spaces:2 Style:italic Page
    Stop
```

where '`SectionLabel`' is the counter that was written out automatically for `\Section`, and '`title`' is whatever information was specified with the '`external`' option.

In effect, this defines the layout of a list that has only one item. Now we see another use for pushing indentation levels: contents items for subsections

may need to be indented, but since they are a separate list on the outer level, we need to push them explicitly to the correct indentation:

```
\DefineExternalItem:SubSection
    file:Contents
    PushIndentLevel Style:roman
    item:right SectionLabel literal:.
        SubSectionLabel Spaces:1 item:stop
    title Spaces:2 Style:italic Page
    Stop
```

Note that a composite label is made here out of the section and subsection numbers.

Implementation. External files are handled in much the same way they are treated in LaTeX: all information is written to the main auxiliary file, and this is loaded at the end of the run, in order to update the other external files.

Writing out titles and such means that these are subject to the usual expansion of `\write`. The LaTeX approach of letting the user put `\protect` commands has proved over time to be too error-prone, so I've decided to inhibit all expansion in titles.

Extendability of Lollipop

For each macro package, the question comes up 'but what if I want something that it cannot do?' The option mechanism of Lolipop can cope with this quite easily. Any option that is undefined is interpreted as a control sequence. Thus the style designer can incorporate arbitrary macros.

For instance, the title pages of *TeX by Topic* (Eijkhout, 1992) have quite elaborate headings, for which I programmed a separate macro `\ChapterHead`, which uses the (automatically generated) macro `\ChapterTitle`.

```
\def\ChapterHead
   {\hbox{ ...
          \PointSize:24 \Style:roman
          \ChapterTitle
    ...}
```

The macro `\Chapter` then uses this `\ChapterHead`:

```
\DefinePageGrid:Chapter
    NextPageGrid:textpage HasTitle:yes
    ...
    ChapterHead
    ... Stop
```

The undefined option '`ChapterHead`' generates a call to the macro `\ChapterHead`.

Goodies

It goes without saying that Lollipop has a sophisticated font selection scheme, a verbatim mode, and other assorted niceties. However, since these facilities are rather pedestrian, if rather useful, I will not discuss them here.

Conclusion

Lollipop is a long, complicated format. An article about it can only give a taste of its philosophy. I hope this piece has given the reader an idea of how macros can be generated automatically, according to the wishes of a style designer. People wanting to use Lollipop can get the software and a user's guide; people wanting to understand it will, for a while, have only this article and the code to go on.

As yet there is no real experience with Lollipop. I have used it myself for two books, but I am the author. I find it very easy to use, but if something goes wrong the errors can be mystifying in the extreme.[5] Error messages are still a major concern. Recall that macros are automatically defined and redefined, by macros that are themselves never explicitly defined. Still, I hope that the dynamic approach will catch enough user mistakes already in the definition stage for this format to be of value to non-TeXnicians wishing to use TeX.

Bibliography

Braams, Johannes, Victor Eijkhout, and Nico Poppelier, "The development of national LaTeX styles", *TUGboat*, **10**(3), pages 401–406, 1989.

Eijkhout, Victor, "A paragraph skip scheme", *TUGboat*, **11**(4), pages 616–619, 1990.

Eijkhout, Victor, *TeX by Topic*, Addison-Wesley, 1992.

Eijkhout, Victor and Andries Lenstra. "The document style designer as a separate entity", *TUGboat*, **12**(1), pages 31–34, 1991.

Perlis, Alan, "Epigrams on Programming", *ACM Sigplan Notices*, **17** (9), pages 7–13, 1982.

[5] And the macros themselves can become pretty big. While debugging, I discovered that TeX will 'only' `\show` the first 1000 characters of a macro...

FoilTEX, A LATEX-like System for Typesetting Foils

James L. Hafner
IBM Research Division
Almaden Research Center, K53/802
650 Harry Road
San Jose, CA 95120-6099
Internet: hafner@almaden.ibm.com

Abstract

FoilTEX is a LATEX-like system for typesetting foils. Its features include simplicity of use, compatibility with LATEX, large sans serif font as default, extra macros to start foils with bold headings and special mechanisms to control the footer and header. There are also facilities incorporated into FoilTEX, when used with compatible drivers, for one-pass multi-color printing. This article describes the basic features and components of FoilTEX.

The system FoilTEX for typesetting foils with TEX is a LATEX-like system designed with a number of goals in mind. The first was simplicity for the user and, in conjunction with this, compatibility with LATEX. It was part of the design plan that one could take an article written in LATEX, delete large sections of the text and with minor modifications make foils from this. One of the special features that was incorporated into FoilTEX, in conjunction with some output drivers, was the ability to get one-pass, multi-color output. In this article, we will describe the basic features of FoilTEX, how some of the features were implemented and some aspects of the use of color.

We should mention that "foils" is an IBMism for transparencies or slides, in the sense of SLITEX. FoilTEX then is intended for preparation of video materials for talks or other presentations (including poster boards). In testing within IBM, FoilTEX has been used to prepare transparencies for overhead projectors, 35mm slides and even materials for teleconferencing.

The FoilTEX Package

The basic FoilTEX package consists of the following files on top of the basic implementation of LATEX.

```
fltplain.tex   foildoc.tex
fltfonts.tex   sampfoil.tex

foils.sty      colordvi.[tex,sty]
foil17.sty     blackdvi.[tex,sty]
foil20.sty
foil25.sty     foilfont.tex
foil30.sty
```

In the first column, the first two files are the heart of FoilTEX. The file `fltplain.tex` defines the basic set of macros (and includes a request to input `latex.tex`) and the second defines all the fonts used by FoilTEX. These are used to build a format (`.fmt`) file.

We remark that FoilTEX was built in the form of a format file primarily because the basic font set used is very different from either LATEX or SLITEX. For example, unlike LATEX, FoilTEX does not load any font smaller than 12pt and unlike SLITEX, it uses scaled 10pt fonts rather than scaled 8pt fonts. It also has fonts available at larger point sizes than both of these other packages. Consequently, to achieve the same effect simply with style files would force almost doubling the preloaded fonts and the *redefining* of a large collection of macros. It was more efficient and simplier to create a new format file.

The next group of files in the first column are the style files that are used with FoilTEX. `foils.sty` is the basic style file used for all foils. The other `.sty` files are used to change default font sizes. See the sections on the `\documentstyle` command and on fonts for more information about these files. There is no `foils.doc` file because the `.sty` file is already relatively well documented.

In the second column, `foildoc.tex` is the source for the documentation which gives more details about the use and installation of FoilTEX. `sampfoil.tex` is a fairly detailed sample foils document.

When FoilTEX was conceived, a suggestion was made to add one-pass color printing capability. This is more related to drivers, but we developed a device independent (but driver dependent) scheme for

doing this. The necessary files are included in this package. The files `colordvi.[tex,sty]` and `blackdvi.[tex,sty]` contain device-independent macros for using color in FoilTEX (or any other TEX). The `.tex` and `.sty` files are identical.

Finally, the file `foilfont.tex` is a FoilTEX file which can be used to test an installation's font availability.

There are other files in the FoilTEX package that we have not listed here. For example, there are system dependent scripts for invoking FoilTEX and some on-line manual or help files. There are also some files for substituting PostScript fonts for the CM fonts or for using some $\mathcal{A}\mathcal{M}\mathcal{S}$-TEX fonts, `msam`, `msbm` and `eufm`, with automatic scaling. These will not be described here.

The \documentstyle Command

To create a FoilTEX document, the user edits a file very much like a LATEX file. Instead of the standard LATEX options specified in the `\documentstyle` command, they would use

`\documentstyle[`*opts*`]{foils}`

Here, the *opts* list can include any style option that one would normally use (and that doesn't corrupt any macros defined by `foils.sty`).

By default, `foils.sty` loads `foil20.sty` and sets up the normal size fonts at 20pt. Analogous to LATEX's `11pt` and `12pt` style options, FoilTEX has `25pt`, `30pt` and `17pt` options. For example, to make normal size at 25pt the command

`\documentstyle[25pt,`*opts*`]{foils}`

will do the trick. Contrary to LATEX, the default `20pt` *is* an acceptable option, though it is redundant.

The Basic Features

The current version of FoilTEX has the following built-in features. The first is that the basic fonts are in large size, approximately 20pt, (so you do not need to do fontsize changing to get large type). The default font is also sans serif as this look better on foils than serif fonts like roman. We have implemented LATEX's font and font size changing commands, relative to this default. More information about fonts and fontsize changing commands can be found in the section "Fonts and Their Sizes".

In spite of the fact that the basic font is sans serif, the numerals and other symbols from the roman font used in math mode are still in the roman font. Thus mathematics will look exactly the same as in LATEX (only larger) but numerals in text will appear in sans serif.

In addition, almost all LATEX macros are available including automatic referencing and citation, table of contents, footnotes, and itemize (which will probably be very popular for foils). The user is not expected to have to do anything to control font types or size changing, except as might be expected in a typical LATEX document.

The following subsections describe a number of additional macros and features that have been defined to make foilmaking easier. In the appendix is a small sample foil document in FoilTEX source and final output form to demonstrate the simplicity and the beauty of the output. In the final few subsections of this section we mention a few of the differences between FoilTEX and LATEX/SLITEX.

The `\maketitle` command. The use of FoilTEX's `\maketitle` command is the same as for LATEX. That is, it reads the contents of `\title{}`, `\author{}`, etc., and produces a titlepage, actually a title foil. The title itself appears centered and down a small space from the top, in a `\Large` bold sans serif font. The author's name with address and date appear under the title, centered and in the `\normalsize` font. If desired, this can be followed by a (necessarily short) abstract with the word "Abstract" appearing in bold and centered above the text of the abstract. See the appendix for a sample. The footer of the title page will contain some special text. See the section on `\MyLogo` for more details.

The new `\foilhead` macro. The first new macro is called `\foilhead`. Its use is described by

`\foilhead[`*length*`]{`*text*`}`

This macro starts a new page and puts *text* in `\large` bold type at the top center of the new page. After the header, a vertical space of approximately 1.0 inch is added providing an automatic cushion between the header and the body of the foil. This vertical space can be adjusted either up or down by putting in the optional argument a TEX *length*. For example, if the body of the foil should sit closer to the header, the command

```
\foilhead[-.5in]{This is the Header}
```

would suffice.

This macro should be used to start any new foil, especially if a new heading is needed. If too much text is intended for a single foil, FoilTEX will do its own page break. This could cause some odd vertical spacing since there is a fair amount of stretchability in vertical glue, particularly in list environments.

This can easily be fixed simply by forcing a page break with an empty `\foilhead{}` command.

The new `\MyLogo` and `\Restriction` macros. Another new pair of macros, `\MyLogo` and `\Restriction`, each of which takes a single argument, are used to control the contents of part of the footline. By default, the footline consists of the contents of `\MyLogo` followed by the contents of `\Restriction` both left justified, with the page number right justified[1]. On the main foils, the default font size is `\tiny`. The contents of these macros can be an empty box as well. By default, `\Restriction` is empty and `\MyLogo` is the phrase "– Typeset by FoilTEX –".

The declarations for these macros would normally be placed in the preamble to the document; i.e., before the `\begin{document}` command. However, these macros can be declared or redeclared at any place in the document. They (and all the other commands that control the footer and header) are sensitive to FoilTEX's output routine, which is essentially unchanged from LaTeX's. Consequently, care must be taken in their placement to be sure they act on the correct pages. In the preamble or immediately after the `\foilhead` command are best. In addition, there are macro switches that can be used to easily turn on or off the logo, without having to do any redeclarations.

`\MyLogo` is really intended for something idiosyncratic to the speaker or his organization. `\Restriction` was included in case you want to have each foil identified for a particular audience. For example, at IBM, we have the option of displaying the IBM logo and words like "Confidential" or "Internal Use Only". The defaults are set in `foils.sty`.

The other three corners of the page. Since the macros `\Restriction` and `\MyLogo` control the bottom left corner of the page, there are other macros for putting text in the other three corners. These are, not surprisingly,

`\rightfooter{`*text*`}`
`\leftheader{`*text*`}`
`\rightheader{`*text*`}`

They each take one argument, the text you want to place in the associated corner of the page. These can also be redeclared within the document with the appropriate attention paid to the output routine.

By default the headers are empty and the lower right footer is just the page number:

```
\rightheader{}
\leftheader{}
\rightfooter{\quad{\sf\thepage}}
```

except on the title page where they are all suppressed. You can easily suppress page numbering by declaring `\rightfooter{}`.

We did not add macros for centering text in the header or footer because we felt this simply add unnecessary clutter to the foils.

New `Theorem` and `Proof` environments. There are a number of (both starred and unstarred) `\newtheorem` environments built in. These are for `Theorem`, `Lemma`, `Corollary`, `Proposition` and `Definition`. Note the uppercased first letter (to avoid possible collisions with user-defined environments of this type). Each must begin and end with `\begin` and `\end` commands as usual. Their text begins with a bold sans serif label like **Theorem** and the content of each is typeset in *slanted sans serif*. The unstarred forms are sequentially numbered and support automatic referencing. The starred forms suppress the numbering and referencing.

There is a `Proof` environment which opens with the word **Proof** and ends with a □. The contents are printed in the normal font.

Mathematics in bold typeface. FoilTEX uses a modified form of LaTeX's font definitions for bold typefaced mathematics. In particular, a `\bf` command in math mode will switch to a bold sans serif font (probably not desirable in mathematics since the rest of mathematics is in serifed fonts). In FoilTEX, LaTeX's `\boldmath` command has been modified also. Here, characters from the roman font are emboldened by switching to the bold roman font (`cmbx` family), not the bold math symbol font as in LaTeX.

To make using bold mathematics easier some new macros have been defined. The first is

`\bm{`*formula*`}`

This takes its argument (within mathematics mode) and replaces it with the emboldened version. Unfortunately, it acts a little funny on characters like summation signs and in super- or subscripts since it reverts to TeX's text style (style T) first. Consequently, this command should be used primarily on individual characters or small parts of formulas.

The second method for getting bold mathematics is a pair of environments

`\begin{boldequation}`
formula (*number*)
`\end{boldequation}`

[1] For the title foil, there is no page number; `\MyLogo` and `\Restriction` are centered and appear in `\footnotesize` font.

```
\begin{boldequation*}
formula
\end{boldequation*}
```

They both set *formula* in bold (except for super- and subscripts). The unstarred form has automatic referencing and is numbered; the starred form inhibits the numbering and referencing.

The limitation on the super- and subscripts not appearing in bold face is strictly to limit the number of fonts loaded by FoilTEX. It was felt this bold math feature would have limited use and so it is not fully supported. It could easily be extended if there is sufficient demand.

List environments. The vertical spacing of items in list environments is controlled by exactly the same mechanisms as in LATEX. We have set the defaults, however, so that at the highest level there is a fair amount of vertical space, but at lower levels this shrinks to nothing. This seemed to produce the best and most pleasing results, at least to the author's personal taste.

This is *not* LATEX. At the heart of FoilTEX is a format file. Consequently, there is usually a system dependent exec (or script or batch program) which calls the main TEX program with the necessary FoilTEX format file, `fltplain.fmt`. Testing showed that users (especially hard-core LATEX users) tended to run LATEX instead out of habit. As a result, a special feature was implemented in which, if LATEX is called on a FoilTEX file, the user is prompted with a warning and given a choice of continuing with some unpredictable consequences or aborting.

Differences with LATEX. One simple difference is that the LATEX command `\em` switches from any unslanted font to *slanted sans serif* and from any slanted font to unslanted sans serif, not to *text italics* and roman, respectively.

Unlike TEX/LATEX, numerals in FoilTEX look different when they are in ordinary text from when they are in math-mode. This means that `12345` in text will print as 12345 and `$12345$` prints as 12345.

Hyphenation has been eliminated from FoilTEX. It was felt that this improves readability. Because of this, FoilTEX might have problems fitting things nicely on a line. Overfull and underfull `\hboxes` might occur more often than in LATEX but the tolerances are set to reduce their frequency. Since the fonts are so large, FoilTEX can be more tolerant of white space without being unaesthetic. If they do occur with no obvious fix, a discretionary hyphen strategically placed can resolve the problem.

Some user's felt that `\raggedright` is preferable for foils. It was decided not to make this the default, but to leave this to the user's discretion.

The following features of LATEX have been disabled in FoilTEX because they seemed unnecessary. They can easily be added if there is sufficient demand: lists of figures, indexing, glossary.

Major differences with SLITEX. There are many differences between SLITEX and FoilTEX. The most glaring feature not supported in FoilTEX is invisible fonts. Also, as indicated in Table 1, `\rm` and `\sf` do what you expect, that is switch to roman and sans serif, respectively. In SLITEX, they reverse roles.

The basic features: future versions. A possible new feature might be an automatically-generated "Summary of the Talk", akin to a table of contents, where the user could tag some of the `\foilhead` macros and have them collected in a special foil following the title foil.

Fonts and Their Sizes

As noted earlier, the default font at `\normalsize` is a sans serif font at size 20pt, unless one of the `[17pt]`, `[25pt]`, or `[30pt]` options have been declared in the `\documentstyle` command. Table 1 shows the control sequences for other accessible text fonts and the name of the font in a sample of its type. These control sequences give the font at the current size. Font size changing commands for each of the normal point size options are described by Table 2. Note that `\bf` and `\sl` yield sans serif fonts, not the usual variations on roman.

Mathematics is also automatically displayed at normal size unless magnified by a size changing declaration. Table 3 describes the font point sizes for TEX's mathematics styles at each of the normal point size options. FoilTEX loads or knows about enough fonts, particularly symbol fonts, that there should never be a discrepancy between the size of text and mathematics at any of the different sizes (unlike LATEX where some fonts at `xxvpt` are actually only 20pt fonts).

Since many of FoilTEX's fonts are not in the standard distribution, and so not available on most systems, the installer will probably have to run METAFONT to generate the necessary files. The file `foilfont.tex` requires a sample of every preloaded or load-on-demand font and so can be used to test an installation's font availability. (Some drivers, like Tom Rokicki's `dvips` program, will generate all the missing fonts just by trying to process this file.)

The LaTeX circle and line fonts have been preloaded at `magstep4` so that small LaTeX pictures should scale naturally to a foil.

Table 1: Available fonts and their names.

command	font names
`\sf`	Sans Serif
`\it`	*Text Italic*
`\sl`	*Slanted Sans Serif*
`\bf`	**Bold Sans Serif**
`\tt`	Typewriter
`\rm`	Roman
`\sc`	SMALL CAPS

Table 2: Type sizes for FoilTEX size-changing commands for the different documentstyle options.

`size/doc-opt`	20pt	17pt	25pt	30pt
`\tiny`	12pt	12pt	12pt	14pt
`\scriptsize`	12pt	12pt	14pt	17pt
`\footnotesize`	14pt	12pt	17pt	20pt
`\small`	17pt	14pt	20pt	25pt
`\normalsize`	20pt	17pt	25pt	30pt
`\large`	25pt	20pt	30pt	36pt
`\Large`	30pt	25pt	36pt	43pt
`\LARGE`	36pt	30pt	43pt	43pt
`\huge`	43pt	36pt	43pt	43pt
`\Huge`	43pt	43pt	43pt	43pt

Table 3: Mathematics type styles and their point sizes at `\normalsize` for the different documentstyle options.

`style/doc-opt`	20pt	17pt	25pt	30pt
D, D',T,T'	20pt	17pt	25pt	30pt
S,S'	14pt	12pt	17pt	20pt
SS,SS'	12pt	12pt	14pt	17pt

Making Color Foils

This feature is still in the development stage and is *very* device-driver dependent. This last problem is regrettable because it severely limits portability, but this cannot be helped at the moment because TeX was not designed with color in mind. In any case, what we have incorporated into FoilTEX are sets of macros which are device independent. They of course use TeX's `\special` command, but do not use any syntax that is dependent on the physical device or output data stream. In this way, it is hoped that more drivers can take advantage of the same set of macros for color printing or display. The only drivers we are aware of that fully support the macros we provide here are Tom Rokicki's `dvips` program (version 5.478 or later) and TeXview on the NeXT. In the next few sections we will describe this implementation of color.

One other comment: these color macros are not necessarily limited to FoilTEX but can run under any other TeX. However there are subtleties about how footlines, headlines, and other special regions of the text will handle the color changes. For very successful use, some macros need to be modified with implied color. We have not tested this explicitly but foresee no special difficulties (provided the driver operates compatibly). The relevant macros in FoilTEX already have these features built in. For example, the footer and header macros wrap everything in `\Black` so colors in the text that cross a page boundary will not affect these regions.

The style file `colordvi.sty` and `dvips`. As we see it, the "right" way to use color in FoilTEX (or other TeX) files is with the `colordvi.sty` file. These macros can be included in FoilTEX, for example, by simply adding `colordvi` to the `\documentstyle` command:

```
\documentstyle[colordvi]{foils}
```

(In TeXs that don't have documentstyles, the appropriate `\input` command will work as well.) This file defines all the color macros using TeX's `\special` command. The internal syntax has forms like

```
\special{color push Red}
Nested Red text.
\special{color pop}

\special{color Blue}
Base color now Blue.
```

depending on whether this is nested color or global color changes (see the section on color macros). Consequently, the driver must be able to recognize the `\special` keyword `color` and process something to the output file that signals the color change, tracking the nesting level, etc. It is also important that the driver be able to track the color state across page boundaries or any other boundary where the output state can change. The driver should ideally also produce output where each page has self-contained color state information, so that pages can be printed in different orders, or by selected pages.

An additional macro in `colordvi.ps` can be used to set the background color. For this macro,

the driver needs to recognize the `\special` keyword `background` and must be able to set the specified background color on the *current* page and remember that color until changed explicitly.

Furthermore, the actual color parameters need to be set in some device dependent way, say with a special prolog file that defines the color `Red` in terms the output device can understand, and in such a way that the parameters are tuned to the particular device. (Each output mechanism uses different color renditions which makes it very difficult to set a universal standard.)

For Rokicki's `dvips`, we have done all of the above. There is a color prolog file which `dvips` includes in its header list whenever it encounters the keywords `color` or `background`. The particular one we wrote has the color parameters tuned to the Tektronix PHASER printer. We added code to `dvips` to track the color history and states during the prescan. In this way, it can initialize the color state on each page of the output file during the final scan. (We should mention that our original code for `dvips` and our original set of macros where greatly improved by Tom Rokicki. We are grateful for his help and for including these features in his driver.)

Finally, we remark that we have used the names `color` and `black` suffixed by `dvi` so as not to conflict with Leslie Lamport's `color.sty` which has become somewhat wide-spread. We chose the suffix `dvi` because it reflects the device independent nature of the macros.

Printing in black/white, with or without `blackdvi.sty`. A FoilTEX (or other TEX) document written with color macros can be printed in black and white in two ways. If the device is a black and white version of a color device (e.g., display or PostScript printer) then it should print in corresponding grey-levels. This is useful since in this way one can get a rough idea of where the colors are changing without using expensive color printing devices. The second option is to replace the call to input `colordvi` with `blackdvi`. This "black" style file turns all the color macros into no-ops, and so will produce normal black/white printing without the user having to ferret out the color commands. Also, most device drivers will simply ignore the color commands and so print in normal black and white.

The color macros: user's viewpoint. There are two kinds of color macros, ones for local color changes to, say, a few words or even one character and one for global color changes. All the color names use a mixed case scheme. There are 68 predefined colors, with names taken primarily from the Crayola 64 crayon box, and one pair of macros for the user to set his own color pattern. More on this extra feature later. Users can browse the file `colordvi.sty` for a list of the predefined color names.

A local color command is in the form

```
\ColorName{this will print in color}
```

As this example shows, this type of command takes one argument which is the text that is to print in the selected color. This can be used for nested color changes since it should restore the original color state when it completes. For example, suppose a user was writing in green and wanted to switch temporarily to red, then blue, back to red and restore green. Here is one way to do this:

```
This text is green but here we are
\Red{switching to red,
\Blue{nesting blue} recovering the
red} and back to original green.
```

In principle the nesting level is unlimited, but it is not advisable to nest too deep lest one loose track of the root color or exceeds the driver's capacity.

The global color command has the form

```
\textColorName
```

This macro takes no arguments and immediately changes the default color from that point on to the specified color. This of course can be overridden globally by another such command or locally by local color commands. For example, expanding on the example above, we might have

```
\textGreen
This text is green but here we are
\Red{switching to red,
\Blue{nesting blue,} recovering the
red} and back to original green.
\textCyan
The text from here on will be cyan
unless \Yellow{locally changed
to yellow}.  Now we are back to cyan.
```

The color commands will even work in math mode and across math mode boundaries. This means that a color state going into math mode will force the mathematics to be set in that color as well. More importantly however, in alignment environments like `tabular` and `eqnarray`, local color commands *cannot* extend beyond the alignment characters.

Because local color commands respect only some environment and deliminator changes besides their own, care must be taken in setting their scope. It is best not to have then stretch too far.

User definable colors. There are two ways for the user to specify colors not already defined. For local

changes, there is the command `\Color` which takes two arguments. The first argument is a quadruple of numbers between zero and one and specifies the intensity of cyan, magenta, yellow and black (CMYK) in that order. The second argument is the text that should appear in the given color. For example, if a user wants the words "this color is pretty" to appear in a color which is 50% cyan, 85% magenta, 40% yellow and 20% black, they would use the command

```
\Color{.5 .85 .4 .2}{this color is
pretty}
```

For global color changes, there is a command `\textColor` which takes one argument, the CMYK quadruple of relative color intensities. For example, to make the default color to be as above, then the command

```
\textColor{.5 .85 .4 .2}
The text from now on will be this
pretty color.
```

will suffice.

If the intended output device does not treat color in CMYK terms, then the device *driver* should convert these values to the device dependent parameters, e.g., RGB.

Setting the background color. There is an additional macro for setting the background color. It takes a single argument, which can either be one of the predefined color names or a quadruple of CMYK values. For example,

```
\background{SkyBlue}
```

or

```
\background{.1 .2 .3 .1}
```

These should appear somewhere on the page (preferably near the beginning) where the background color is to change. The background should stay this color until explicitly changed by another such command. It should be remembered that the placement of this is sensitive to the output routine.

Default color regions in FoilTEX. When `colordvi.sty` is loaded by FoilTEX, the default text color is black. The footline and headline get defaulted color values as well. So, the contents of `\MyLogo`, `\Restriction`, `\rightfooter` (the page number by default), `\leftheader` and `\rightheader` will all appear in Black.

Overriding the color selection for any of these regions of the text can be done by using *local* color commands in their declarations. For example, a very sensitive talk requiring the words "Need-To-Know" in red would use the declaration

```
\Restriction{\Red{Need-To-Know}}
```

Installing \FoilTeX

Because installations of TEX/LATEX differ so much from system to system and even within systems, the installation instructions here are mostly just an outline of the general procedure and system requirements.

To install FoilTEX, the requirements are:

- the TEX program, including a version of `INITEX`;
- LATEX, any version after Nov. 89 (The concern here is access to the font metrics for `lcircle10` and `lcirclew10`, as opposed to `circle10` and `circlew10`. The installer can modify `fltfonts.tex` if necessary.); and
- the METAFONT program and related tools to generate the necessary fonts (FoilTEX uses essentially all the basic CM fonts and some additional LATEX fonts but at magnifications equivalent to `\magstep6`, 7 and 8.).

The installation of FoilTEX then involves

- generating a `fltplain.fmt` format file by running `initex` (and installing this in the appropriate location for the system);
- testing font availability by running TEX with this format against `foilfont.tex` and trying to print this;
- generating all the missing fonts; and
- installing the various style, macro, script and on-line help files in the appropriate location for the system.

Acknowledgements and Requests

We would like to thank and acknowledge the following people in IBM for their great assistance in helping to put FoilTEX together: Katherin Hitchcock, Myron Flickner, Ekkehard Blanz, Melanie Fulgham, Peter Haas, Rocky Bernstein and the many users who contributed their constructive comments on the early test versions within IBM.

A special thanks goes to Tom Rokicki for implementing our color setup in his driver and another to Sheri Gish of IBM for asking the right (or was it wrong?) question that got this project started.

FoilTEX is intended to be easy to use, useful and to produce beautiful foils. Consequently, the author welcomes any comments or suggestions.

Sample Foils

Below is source for a short two page sample foil that demonstrates most of the features of FoilTEX, followed by a facsimile of the output from this source.

```
%%%%%% First we load the correct style file
\documentstyle{foils}
%%%%%% This first section is for a title page; it is typical LaTeX
\title{Rock protocols for binary Quarries}
%
\author{Fred Flintstone\\
Rock Quarry Research Center}
\date{\today}
%%%%%% This next command controls part of the footline.
%%%%%% Note the ``FoilTeX'' logo will print automatically.
%\yourlogo{-- Typeset by \FoilTeX\ --}
\Restriction{TUG Use Only}
%
\begin{document}
\maketitle                          %
\begin{abstract} This is where an abstract might go.\end{abstract}
%%%%%% This next command starts a new foil with header.
\foilhead{Variability of Rock Quality}
%
What can we prove using only marble rocks?
%%%%%% Itemize, mathematics, auto-referencing and footnotes are built-in.
\begin{itemize}
\item $\Omega(t^2)$ rocks needed \cite{rocky}\footnote{What's that?}.
\item Worst case structure uses
      \begin{equation} \label{equation}
                O(n+t\sqrt{t})
      \end{equation}
\end{itemize}
%%%%%% Here is a sample theorem with proof.
\begin{Theorem} Everything you know about rocks is false.
\end{Theorem}
%
\begin{Proof} The proof is obvious from equation (\ref{equation}).
\end{Proof}
%%%%%% Bibliographies work even with BibTeX.
\begin{thebibliography}{99}
%
\bibitem{rocky} Rocky and Bullwinkle, Open problems, in {\sl Mr.
Know-it-all's Rock Encyclopedia}.
%
\end{thebibliography}
\end{document}
```

Rock protocols for binary Quarries

Fred Flintstone
Rock Quarry Research Center

September 2, 1992

Abstract

This is where an abstract might go.

– Typeset by FoilTEX – TUG Use Only

Variability of Rock Quality

What can we prove using only marble rocks?

- $\Omega(t^2)$ rocks needed [1][1].
- Worst case structure uses

$$O(n + t\sqrt{t}) \tag{1}$$

Theorem 1. *Everything you know about rocks is false.*

Proof. The proof is obvious from equation (1). □

References

[1] Rocky and Bullwinkle, Open Problems, in *Mr. Know-it-all's Rock Encyclopedia*.

[1] What's that?

– Typeset by FoilTEX – TUG Use Only 1

Typesetting a Magazine the Easy Way

Peter Abbott
Information Systems
Aston University
Aston Triangle
Birmingham B4 7ET
United Kingdom
Phone: +44 (0)21 359 5492;
FAX: +44 (0)21 359 6158
Internet: p.abbott@aston.ac.uk

Abstract

'THE CORNISHMAN' is a quarterly publication in A5 format of some 56 pages per issue. Text is mostly set in two columns with photographs and adverts which may be column width or span two columns (at the top or bottom of a page). During makeup it may be necessary to move photographs from the top to the bottom of a page or vice versa. Changes have been made to the basic style file to obtain the required text size and baseline spacing. Modifications to footnotes coupled with multicolumn style achieves photographs at the bottom and the use of the figure* environment at the top. Single column photographs are more difficult to place. A number of macros have been written to automate the process as much as possible.

Introduction

I edit and typeset a quarterly magazine, the contents of which is text, display-adverts and photographs (images). Currently the print run is 1750 copies of between 52 and 64 A5 pages. The copydate is normally one calendar month before publication.

I became the editor in the summer of 1989 when the magazine was approximately 12–16 weeks late with each issue, was only 44 pages and naturally only tolerated by its readers. The editor submitted handwritten or typed manuscripts to the typesetter/printer and had very little control over the layout and presentation.

I agreed to take the editorship provided that:

- I could print the magazine in house,
  and
- control the style and presentation.

In return I agreed with the GWR Ltd Board (who publish 'The Cornishman') to:

- eliminate the delays and **PUBLISH ON TIME** (To date I have not missed a single deadline and the March issue was 64 pages.);
- increase the content and quality;
- increase the photographic coverage and quality;
- encourage the volunteers; and
- make the publication more attractive to the reader, and generally promote our Railway.

I have been using LaTeX for some time and this seemed suitable for my needs. In 1989 I had a Mac IIci (since then I have acquired a Mac IIfx). One of the major problems I have encountered is storage of data (especially images) and have found an optical (multiple write) disc with 500mb cartridges a slow but essential storage medium. Of course PostScript output has made my life easier (from my point of view) in that proof copies can be printed on a 300 dpi laserwriter with camera ready copy on the Linotronic at 1270 dpi via a RIP.

I use LaTeX for convenience and I am currently using OzTeX version 1.41.

Basic Format

The magazine (called 'The Cornishman') is A5 sided two column format.

I normally require photographs to be either single column or full width with captions and acknowledgments. I dislike (continued on page ...) and therefore once an article is started it must be contiguous. This sometimes creates real problems in making up the final copy. All the examples used are from issue No. 40 which was published on 1st March 1992.

Processing Text

The magazine was produced in two column format when under the control of the previous editor and I therefore decided to continue with this format. The options available are `\twocolumn` or the `multicols` style. I selected `multicols` and my standard processing file is

```
\documentstyle[multicol,cornish,shadow%
,array,ifthen]{article}
\newcommand{\status}{%
\setcounter{omitpic}{1}}
\begin{document}
\status\clearpage\input{:atext:class28}
\end{document}
```

I tried using `\include` but occasionally ran out of space. It is a trivial matter to create a file in the subdirectory `atext` called `aaorder` containing all the input lines for an issue in the order in which they should be processed. I can then process part of the magazine or the whole issue (page references do not cause problems with this method). A value for `omitpic` of `0` draws a box where an image will appear so that I can judge the overall effect before including an image.

Changes to standard macros/styles

The normal 10pt size is too large for an A5 magazine so I selected part of `article.sty` and included it in `cornish.sty`

```
\def\@normalsize{\@setsize\normalsize{12pt}%
\xpt\@xpt
\abovedisplayskip 10pt plus2pt minus5pt%
\belowdisplayskip \abovedisplayskip
\abovedisplayshortskip \z@ plus3pt%
\belowdisplayshortskip 6pt plus3pt
minus3pt\let\@listi\@listI}
\def\small{\@setsize\small{11pt}\ixpt\@ixpt
\abovedisplayskip 8.5pt plus 3pt minus 4pt%
\belowdisplayskip \abovedisplayskip
\abovedisplayshortskip \z@ plus2pt%
\belowdisplayshortskip 4pt plus2pt minus 2pt
\def\@listi{\leftmargin\leftmargini %
\topsep 4pt plus 2pt minus 2pt\parsep 2pt
plus 1pt minus 1pt
\itemsep \parsep}}
\def\footnotesize{\@setsize%
\footnotesize{9.5pt}\viiipt\@viiipt
\abovedisplayskip 6pt plus 2pt minus 4pt%
\belowdisplayskip \abovedisplayskip
\abovedisplayshortskip \z@ plus 1pt%
\belowdisplayshortskip 3pt plus 1pt minus 2pt
\def\@listi{\leftmargin\leftmargini %
\topsep 3pt plus 1pt minus 1pt\parsep 2pt
plus 1pt minus 1pt
\itemsep \parsep}}
\def\scriptsize{\@setsize%
\scriptsize{8pt}\viipt\@viipt}
\def\tiny{\@setsize\tiny{6pt}\vpt\@vpt}
\def\large<{\@setsize\large{14pt}\xiipt\@xiipt}
\def\Large{\@setsize\Large{18pt}\xivpt\@xivpt}
\def\LARGE{\@setsize\LARGE{22pt}%
                    \xviipt\@xviipt}
\def\huge{\@setsize\huge{25pt}\xxpt\@xxpt}
\def\Huge{\@setsize\Huge{30pt}\xxvpt\@xxvpt}
```

and modified some of the code (only the modified lines are shown)

```
\def\@normalsize{\@setsize%
\normalsize{9.68pt}\ixpt\@ixpt
\def\small{\@setsize\small{9pt}\viiipt\@viiipt
\def\tiny{\@setsize\tiny{7pt}\vipt\@vipt}
\def\large{\@setsize\large{11pt}\xpt\@xpt}
\def\Large{\@setsize\Large{14pt}\xiipt\@xiipt}
```

With an A5 format some of the spacing provided by environment changes in LaTeX are unacceptable and therefore I have found it necessary to modify a number of parameters. This gives me greater control over page layout but for larger page sizes the default values are acceptable. I have not included these parameters here as they are my preferences and were arrived at by trial and error. I worked on the principle that although white space can be used to improve output, in small formats such as A5, the parameter values are too big. There are two parameters which must be mentioned, `columnwidth` and `textwidth` and they are `59mm` and `122.5mm`. The reason for quoting these two values becomes obvious when you see examples quoted.

My Additions

I decided to adopt a house style for the name of the magazine 'The Cornishman' and currently have the following macro defined.

```
\newcommand{\magazine}%
{{\bf\sf\mbox{'The Cornishman'}}}
```

Originally I did not have the `mbox` but after a number of issues where the word 'Cornishman' was split over two lines of output text I added the `\mbox`.

Note that the use of the new font selection scheme makes life much simpler. Before the arrival of NFSS, inserting `\sf` meant that the type size in use was lost. I found this most frustrating and consequently how useful NFSS has proved.

Perhaps I should say at this point that the macros have been in a state of development since 1989 and are not yet finalised, and I often make minor adjustments to make my life easier. In fact, since publishing issue 40, I have made a number of changes to the macros and installed a new version of `multicols` which Frank Mittelbach asked me to

beta test. As a result of comments received from issue No. 41, the inter-paragraph space has been removed and `parindent` changed from zero to `5mm`.

The previous editor (or printer) selected a variety of styles for identifying articles. There was no consistency in that sometimes the writer was credited and sometimes not. Some were reversed white on black. Some titles were UPPER CASE but most were mixed.

I selected a style of a greybox with a black line top and bottom, either full width or column width and the macros are therefore

```
\newcommand{\bstitle}[5]{{\Large%
\addcontentsline{toc}{subsection}{#1}}%
\nocont{#1}{#2}{#3}{#4}{#5}}%

\newcommand{\nocont}[5]{%
\setlength{\unitlength}{1mm}%
\begin{picture}(#4,11)%
\put(0,0){\special{#5.ps}}%
\put(0,0){\framebox(#4,0){\ }}%
\put(0,11){\framebox(#4,0){\ }}%
\put(0,0){\makebox(#4,11)[l]{%
\sf\bf\hspace*{1mm} #1}}%
\put(0,0){\makebox(#4,11)[r]{%
\sf\bf {\rm\it #2 \/  \sf%
\bf\  #3\hspace*{1mm} }}}%
\end{picture}}%

\newcommand{\dtitle}[2]{\bstitle{#1}{by}%
{#2}{122.5}{greylbox}}%
\newcommand{\stitle}[2]{\bstitle{#1}{by}%
{#2}{59}{greybox}}%
\newcommand{\dblk}[1]{\bstitle{#1}{\ }%
{\ }{122.5}{greylbox}}%
\newcommand{\sblk}[1]{\bstitle{#1}{\ }%
{\ }{59}{greybox}}%
```

where `greybox` and `greyboxl` are column width and textwidth respectively and `greybox` is

```
72 2.54 div 72 2.54 div scale
% units are now cm instead of big points

newpath 0 -.0 moveto 5.9 0 rlineto
                      0 1.1 rlineto
                     -5.9 0 rlineto

closepath        % complete rectangle
gsave            % save current path
.75 setgray      % use very light shading
                 % on LaserWriter
                 % A(0=black, 1=white)
fill             % paint interior of rectangle
grestore         % restore current path
```

The macros give titles either column or text width, with or without credit as appropriate.

Using Macros which Exist

The magazine contains a table of contents and I decided that I could use `\tableofcontents` provided that I removed the heading 'Contents' which is automatically output. After using this for several issues I decided that the white space to the left of the title produced when using `tableofcontents` was spoiling the appearance of the column and I never numbered articles.

I preferred the contents to neatly fill the column widthwise. I therefore extracted the relevant code from `article.sty` and inserted into `cornish.sty`

```
\def\tableofcontents{%\section*{CONTENTS%
        %\@mkboth{CONTENTS}{CONTENTS}}
% NOTE THE % IN THE PREVIOUS TWO LINES
\@starttoc{toc}}
\def\contentsline#1{\csname l@#1\endcsname}

\def\@dottedtocline#1#2#3#4#5{%
\ifnum #1>\c@tocdepth \else
  \vskip \z@ plus .2pt
  {%\leftskip #2\relax
 % NOTE THE % IN THE PREVIOUS LINE
 \rightskip \@tocrmarg \parfillskip -\rightskip
   % \parindent #2\relax
% NOTE THE % IN THE PREVIOUS LINE
 \@afterindenttrue
   \interlinepenalty\@M
   \leavevmode
   \@tempdima #3\relax \advance\leftskip
\@tempdima \hbox{}\hskip
-\leftskip
    #4\nobreak\leaders\hbox{$\m@th \mkern %
\@dotsep mu.\mkern \@dotsep
     mu$}\hfill \nobreak \hbox to\@pnumwidth{%
   \hfil\rm #5}\par}\fi}
```

The major problem revolves around dealing with images (most, but not all, are photographs). In fact, a display advert or other material can be treated as an image.

A photograph will be either full width or column width with a caption in *italics* and the credit to the photographer right justified.

I therefore wrote a macro using the one quoted on page 106 of *The TEXbook*.

```
% see page 106 of TeXbook
\def\photo#1{{\parskip=0pt{\unskip\nobreak%
\hfil\penalty50\hskip1em\hbox{}\nobreak\hfil
#1\parfillskip=0pt\finalhyphendemerits=0\par}}}

\newcommand{\pict}[5]{\ponly{#1}{#2}{#3}%
\newline {\it #4\photo{Photo: #5}}}%
\newcommand{\ponly}[3]{%
\setlength{\unitlength}{1mm}%
\begin{picture}(#2,#3)%
```

```
\put(0,0){\ifthenelse{\value{omitpic}=0}{%
\framebox(#2,#3){{\large #3mm
#1}}}{\makebox(#2,0)[lb]{%
\special{epsf=epsfiles:#1}}}}%
\end{picture}}%
```

There is one vital ingredient in the above code, the assumption that the `.eps` file is stored such that the bottom left corner of the image is represented by the coordinates `(0,0)`. To be able to put two single column pictures side by side is easy.

```
\dtitle{The 1991 Sponsored Trackbed Walk}%
{Garry Owen}
\parbox[b]{59mm}{%
\pict{t1.eps}{59}{45}{The walkers assemble
on Race Course Platform.}%
{Steve Standbridge}}\hfill%
\parbox[b]{59mm}{
\pict{t4.eps}{59}{45}{Gotherington Signal
Box.}{Steve Standbridge}}\begin{multicols}{2}
The sponsored Trackbed Walk took \ldots
```

Following presentation of a similar paper to the DANTE meeting in Hamburg in March 1992, I experimented with Frank Mittelbach's beta test version of `multicols`. The code above then becomes (and in fact at the start of a section could always have been

```
\dtitle{The 1991 Sponsored Trackbed Walk}%
{Garry Owen}
\begin{multicols}{2}
\pict{t1.eps}{59}{45}{The walkers assemble
on Race Course Platform.}%
{Steve Standbridge}

\pict{t4.eps}{59}{45}{Gotherington Signal
Box.}{Steve Standbridge}
\end{multicols}

\begin{multicols}{2}
The sponsored Trackbed Walk took \ldots
```

If the text for the two photographs does not occupy the same amount of vertical space, then an adjustment to the height of the photograph is required. This is when the ability to draw a box to represent the image is really useful.

I then needed to be able to put a picture at the bottom of a page and decided to use the facilities of footnotes. I extracted the following from `latex.tex`

```
\long\def\@footnotetext#1{\insert\footins{%
   \footnotesize
   \interlinepenalty\interfootnotelinepenalty
   \splittopskip\footnotesep
   \splitmaxdepth \dp\strutbox
   \floatingpenalty \@MM
   \hsize\columnwidth \@parboxrestore
   \edef\@currentlabel{%
      \csname p@footnote\endcsname%
   \@thefnmark}\@makefntext
    {\rule{\z@}{\footnotesep}\ignorespaces
      #1\strut}}}
```

I then modified the code and produced my own macro `\footpict` as follows:

```
\def\footpict{\@footpict}
\long\def\@footpict#1{\insert\footins{%
    \hsize\columnwidth \@parboxrestore
    {#1}}}
\let\footnoterule\@empty
```

This means that in the source code

```
The train continued to run along our line
until 7th September 1962 when it was
transferred to the Bromsgrove line
(see \magazine\ No.\ 38).

\footpict{\pict{eddieb.eps}{122.5}{78}%
{GWR \locono{5080} 'Defiant' storming up
the bank towards Greet Tunnel.}{Dennis Bryan}}

When the Gloucestershire Warwickshire Railway
was formed and \ldots
```

`\footpict{}` is inserted at the first convenient paragraph break on the page where it is desired to have a photo at the bottom. This works in both single column and `multicol` mode.

Placing two photos at the bottom of the page is a trivial problem; simply insert the double `parbox` code inside `footpict`. The beta test version of `multicols` again provides for the use of multicols within the `footpict` macro. Again, if the text demands different height values, the use of `omitpic` with a value of `0` makes life easy. It is also worth noting that the width value within `\pict` is only required when images are omitted. It would be possible to use a fixed value, but although I have tried it, I do not find it a problem to specify the width of an image and on occasions it is helpful.

Likewise, photos at the top of a page (as long as they cross both columns) are a simple matter. They can be either one image across both columns or two images, each being a single column width and using the `parbox` method. Again, variable heights will work.

When the following code:

```
\begin{figure*}
% image code inserted here
\end{figure*}
```

is inserted in the text at a suitable paragraph break on page `n`, then the image will appear at the top of page `n+1`.

An image at the top or bottom of a column is slightly more difficult and involves setting the item

somewhere in the column, noting where the column break occurs and then moving it to the correct place. It will probably need the following code as well

```
{\parfillskip=0pt\par}
```

to ensure that the preceding column to the image does not have a false paragraph break when none is intended. Therefore, I have a wish list for an improvement to `multicol`. I would like to be able to float a single column image to the top or bottom of a column. Frank Mittelbach is aware of my wish so all I can say is wait for future developments.

I must reiterate that images need not be photographs and, in fact, all of the display adverts used in my magazine are created using the same code as `ponly`.

Conclusions

I will admit to one failing, the cover(s) and center page are **NOT** produced in LaTeX. This is one case where page layout software on the Mac has a distinct advantage. Frank Mittelbach has admitted that it is unlikely that `multicols` will be able to replace my `footpict`. I know how to manipulate single column floats, the only penalty being an increase in the processing time.

Is this method successful? We (my Assistant Editor, Audie Baker, and I) spend around 100 to 120 hours of time to produce an issue of 60 – 64 A5 pages.

The answer must be yes as now the magazine is always on time and is enjoyed by **ALL** the readers. We usually have to hold material over for the next issue and there are always more than enough photographs for each issue. I see magazines produced by other DTP methods, and (I know I am biased) TeX and LaTeX are still far and away the best for typsetting beautiful text which in fact, apart from fractions (I naturally have written a macro for fractions which appear pleasing to the eye at `normalsize`), has no mathematics included.

I have learned a lot in the past three years on image processing and how to include images in printed material. The basic lesson is that large amounts of storage are required but even more important is **FAST** processing. I have just invested in a SUN IPX workstation with an additional 1.2 gigabyte disc to reduce the overall processing time.

Postscript

A number of changes have occured since issue 40. For issue 42 (Autumn 1992), photographs were stored in compress TIFF format and conerted to PostScript on the fly by DVIPS. This method reduces the disk space requirements for each photograph by more than 60%. The PostScript file used to create the film for plate making for issue 42 was almost 150 megabytes.

Developments continue and it is hoped that for issue 43 the width parameter in `\ponly` will be made redundant. Page imposition will, I hope, be changed to A3 format with each A3 page made up of 4 A5 pages with heads set to the centre. Linotronic output then be suitable for plate making without manual make-up.

My style file, `cornish.sty` will shortly be available in the archives. They will, of course, require modificaton dependent upon the `dvi` to output device program that the user employs. The file also includes my parameter settings; these are

I will always be happy to answer queries about CORNISH.STY but would prefer them to be sent to the `uktex@tex.ac.uk`, as I am one of the reviewers of that list.

Acknowledgements

I must acknowledge the help given by members of the Aston Archive Group. They have made numerous suggestions for improving the magazine and unravelled the complexities of some of the existing `parameters` and `macros`. Finally, I would like to thank my Assistant Editor, Audie Baker, whose speciality is photo composition and arrangement.

Using TEX for a Publications Database

Mimi Burbank and Donna Burnette
Supercomputer Computations Research Institute
Florida State University
Tallahassee, FL 32306-4052
Internet: mimi@scri.fsu.edu; donna@scri.fsu.edu

Abstract

This article explains the impetus and chronology associated with the use of plain TEX at the Supercomputer Computations Research Institute (SCRI) to produce multiple and varied reports, and the evolutionary process of the SCRI "publications database" into its current state. Why would one use TEX for a database? Our reasons were simple: all of our publications were done in TEX, and the versatility of TEX as a programming language made it ideal. We have evolved from using a single file using an \halign to a series of macros which utilize one input file to produce a wide array of output formats according to preset information/design criteria. The reporting process is now quite complex though text entry remains the same — inputting information into only one file!

Introduction

The Supercomputer Computations Research Institute (SCRI) is an interdisciplinary program set up to do research in computational science, train researchers from various academic disciplines and provide them access to a supercomputer. It is a cooperative venture between Florida State University (FSU), the U.S. Department of Energy's Office of Energy Research, the State of Florida, and several computer manufacturers. The SCRI base consists of a 50-plus member group of application research scientists, technicians, software specialists and support personnel. However, this group is supplemented by various numbers of long-term visitors from other universities, collaborating faculty members from other FSU departments and graduate and undergraduate students.

During 1985 – 1986 approximately 94 publications were produced using TEX, which upon looking back, were reminiscent of medieval times (scissors, glue, copiers, black spots, white-out, etc.). At this time, there was no resident TEXpert — only a single person trying to learn and use TEX. During 1986, a computer science student was hired part-time to assist in manuscript preparation, and at this time we became officially responsible for the reporting of publications at monthly and annual intervals. There was no suitable database on-line, and fiscal records were maintained in a variety of Macintosh-related utilities. Since all of our publications were done in TEX, it seemed time- and cost-effective to utilize the programming capabilities of TEX.

We shall try to explain the underlying TEX concepts involved in making our database work. While reading through the details of our setup, it is important to realize that this concept could easily be applied to any collection of information that is reiterated periodically with the only difficulty being the design of the report document itself and the ability to turn on/off the appropriate commands to feed it accurate information.

Pre-Database Publications

With very little experience using TEX, we started out with a three-column design for our publications report, using a variety of fonts to accentuate the different areas required — preprint number, month/year, title, author, and where submitted or published. An example of the code used at that time is:

```
%
\halign{%
  \hbox{\vtop{\hsize=2in\raggedright #}}
   \hfil \quad
   &  {\hbox{\hsize=1in#}}\hfil\quad
   & \hbox{\vtop{\noindent\hsize=4in
    \raggedright #}}  \hfil\cr
{\bf FSU-SCRI-85-01\break}
Bhanot, Duke \& Salvador
& 4/85
&FRACTALS AND INTERPOLATING
DIMENSIONS\cr
\noalign{\vskip1.3ex}
&& {\it Published Phys. Lett. B.},
```

```
Vol. 165B, 12/26/85,
pp. 355--360.}}\cr
\noalign{\vskip3ex}
{\bf FSU-SCRI-85-02\break}
Hasenfratz, A. and  P. Hasenfratz
& 4/85
& LATTICE GAUGE THEORIES\cr
\noalign{\vskip1.3ex}
&& {\it Published in Ann. Rev. Nucl.
Part. Sci., Vol. 35, 1985, pp. 559
--604.}\cr
}}
```

Modification of this data was an exercise in torture — readability was almost zero and time lost debugging forgotten ampersands, \cr's, etc. was commonplace and costly. Excerpts from this file were then copied into other files to represent partial listings that met certain criteria. Our facility's publication rate quickly made manually sorting through this list according to *any* criteria a cumbersome task. We decided to delve into TEX's programming abilities and see if there was a way to automate this process. In 1987, we hired a programmer and officially began maintaining a "real" database, from which we ran monthly reports, fiscal reports and one cumulative report from 1985 onward.

Primitive Database Design

Having learned how to use temporary storage boxes (i.e., \setbox) we decided that this was a place to start our research into TEX's abilities/inabilities. We categorized all of our preprint information and put each item (author, title, subject, date, journal, volume, etc.) into its own box. A sample preprint entry at this stage of our database follows.

```
\num{01}
\date 1/92
\author{D. Burnette}
\title{Using \TeX\ for a
        Publications Database}
\journal{Submitted to TUGboat}
\endref

\num{02}
\date 1/92
\author{Unknown}
\title{This is a Junky Title}
\journal{Published in the Journal of
        Unknown Works}
\volume{5}
\page{1200--1204}
\endref
```

With the addition of a command to re-initialize the contents of the boxes between preprint entries, we had a legible list of preprints which we could output in its entirety at any time. Though there should be a less painstaking method to accomplish this task, the code/command we use for initialization (\resetvars) is defined below. The additional box \dummy is used as a mechanism for discarding information that has already been processed or is otherwise unused. The need for this command arose because of the occasional omission of a field by one of the many people entering information into the publist*nn*.dbf file, which allowed information from one entry to show up in the output of a consecutive item, and we preferred to have an empty field rather than an erroneous field.

```
\newbox\dummy
\def\resetvars{%
%       reset the ALL variables.
%
\ifvoid\refnum{}
\else\global\setbox\dummy=%
   \hbox{\unhbox\refnum}
\fi%
.
.
.
\ifvoid\refextra{}
\else\global\setbox\dummy=%
   \hbox{\unhbox\refextra}
\fi%
}%
%
```

Each database entry contains some combination of the above commands, based on the type of preprint it is and the information we want to maintain, but they are all delimited by the command \endref which does all of the work.

```
\global\def\endref{%
\hsize=7truein\parindent=0pt
  \baselineskip=12.4pt \parskip=6pt%
\if\currentptype\p
 \vtop{\line{\okbreak%
  \vtop{\hsize=2in\noindent\raggedright%
    \ifvoid\refnum{}%
    \else\bf FSU-SCRI-\unhbox\refyear%
    \unhbox\refatype-%
    \unhbox\refnum\break%
    \fi%
   \unhbox\refauthor\par
   \unhbox\refsubject}\hfill%
   \hbox to .5truein{\hfill
    \unhbox\refdate\hfill}%
```

```
    \hfill\vtop{\parindent=0pt
   \hsize=4in\raggedright%
    \ifvoid\reftitle{}%
     \else\unhbox\reftitle\par%
     \fi%
    \ifvoid\refjournal{%
     \ifvoid\refbook{}%
     \else In: \unhbox\refbook%
     \fi%
    \ifvoid\refeditor{}%
     \else,\ \unhbox\refeditor, ed.%
     \fi%
    \ifvoid\refeditors{}%
     \else,\ \unhbox\refeditors, eds.%
     \fi%
    \ifvoid\refvolume{}%
     \else,\ \unhbox\refvolume%
     \fi%
    \ifvoid\refpage{}%
     \else,\ \unhbox\refpage%
     \fi%
    \ifvoid\refpublisher{}%
     \else,\ (\unhbox\refpublisher%
     \fi%
    \ifvoid\refpubyear{).}%
     \else,\ \unhbox\refpubyear).%
     \fi%
     }%
 \else\unhbox\refstatus\unhbox\refjournal%
  \ifvoid\refvolume{}%
   \else,\ \unhbox\refvolume%
   \fi%
  \ifvoid\refpage{}%
   \else,\ \unhbox\refpage%
   \fi%
  \ifvoid\refpubyear{.}%
   \else\ (\unhbox\refpubyear).%
   \fi%
  \fi%
  \ifvoid\refextra{}%
   \else\ \unhbox\refextra.%
   \fi%
   }}%
\bigskip
}\fi%
}
```

As you can see, we have replaced our \halign scheme with \hboxes and \vboxes with defined dimensions.

Defining Our Criteria

Our next task was to examine each record (\num...\endref), set up logical collections of records, and then mark them in some way. In 1989 our funding agency had expressed an interest in a regular listing of our published papers as well as a periodic report of the status of the publications during the reporting period. From that suggestion, we decided that obtaining a listing of papers in any one of the various steps toward publication might also be beneficial. We found that there were three steps in the publication process (submitted, accepted, and published). We also found that we had papers that were submissions to conference proceedings, and another set that were technical reports but otherwise unpublished. We assigned each record a \papertype according to its category and chose the letters \s, \a, \p, \c, and \t as their designations.

Beyond this collection, we also found it helpful to know how much publishing each of our authors have done (individually and as a group). Due to the multidisciplinary/international nature of the institute we have authors whose publications appear in more than one subject area.

We have a group of scientists that more or less make up the core of SCRI. These scientists invite scientists in their field to collaborate with them on projects. Quite often these visitors publish papers on their collaboration while they are here or after they have left. These papers (\v) are added to our database.

There are projects off-site that request allocations of supercomputer time which are granted by our funding agency provided they agree to supply us with the publications generated by their results. Once a year we solicit the external users of the supercomputer for their publications and also add them (\e) to our database.

We found that assigning an \authortype to each preprint would enable us to generate a report of work done by any particular type of author should the need arise and chose the letters 'e', 'v', and none (the default) as their designations. The default \authortype is used for members of the core group of SCRI scientists.

Assigning the Criteria Commands

We created a set of commands to hold the characters which would represent each paper category (\s, \a, \p, \z, \t, \e, \v) and a second set of commands to associate those conditions with the record 'type' in the database (\papertype corresponded to \currentptype, and \authortype corresponded to \currentatype) for comparison at

run-time. \z was substituted for \c (to avoid conflict with TEX's cedilla) for conferences papers.

```
%
\global\def\a{A}  % to appear
\global\def\e{E}  % external
\global\def\p{P}  % published
\global\def\s{S}  % submitted
\global\def\t{T}  % technical
\global\def\v{V}  % visitor
\global\def\z{C}  % conf. proc.
%
```

Further refinement resulted in the following definitions, which include the introductory remarks regarding publication status. This was done to maintain uniformity of entries.

```
\global\def\published{\papertype{P}%
   \global\setbox\refstatus=\hbox{\it
    Published in: }}
\global\def\submitted{\papertype{S}
   \global\setbox\refstatus=\hbox{\it
    Submitted to: }}
\global\def\accepted{\papertype{A}
   \global\setbox\refstatus=\hbox{\it
    To appear in: }}
\global\def\technical{\papertype{T}
    \tech{SCRI Technical Report}}
\global\def\conf{\papertype{C}}
.
.
```

Using the above samples, database entries quickly changed to:

```
% SCRI author / submitted paper
\num{01}
\date 1/92
\submitted
\author{D. Burnette}
\title{Using \TeX\ for a
        Publications Database}
\journal{Submitted to the TUGboat}
\endref

% SCRI visitor / published paper
\num{02}
\date 1/92
\visitor
\published
\author{Unknown}
\title{This is a Junky Title}
\journal{Published in the Journal of
        Unknown Works}
\volume{5}
\page{1200--1204}
\pubyear{1992}
\endref

% External Author / accepted paper
\num{03}
\date 1/92
\external
\accepted
\author{Unknown}
\title{Another Junky Title}
\journal{Published in the Journal of
        Something Else}
\endref
```

Finally, we incorporated conditional statements into our definition of \endref to test for various types of entries. It is important to note the conditional statement \if\currentptype\p takes the current value of \currentptype and compares it to the current value of \p. From the previous definitions, we know that \p should always be "P". The value of \currentptype is set in the record entry by \papertype{<value>}. If <value> = "P" also, then this conditional equates to TRUE and the various details about the preprint are unboxed according to the macro. Otherwise, the information in this record is not needed and no output is generated by this entry. It is fairly easy to see that if you include a conditional statement for each type of record that you need to search for and vary the boxes that are utilized according to the type of entry involved, overall control of the output generated is fairly simple to tailor to your individual needs. We have constantly rehashed our use of conditional statements to remove unnecessary comparisons and to cut down on the confusion caused by nesting too many conditional statements, each of which has to be evaluated anyway (tabbing can only do so much!).

Current Status

The refinement of the file has progressed to its present state in which the following information is maintained:

```
\num{<sequential = {1,2,3,...,N}>}
\date mo/yr
\author{<required>}
\external {or  \visitor}
\submitted {or \accepted,
    \conf, \published, or \technical}
\title{<required>}
\volume{<if published>}
\page{<if published>}
\pubyear{<if published>}
\extra{<optional comment>}
\reprinttrue
```

```
   \journal-code <or \proc or \tech>
   \SUBJECT-AREA-CODE
   \AUTHOR1-CODE\AUTHOR2-CODE
\endref
```

Our database files are called `publist`*nn*`.dbf` with *nn* being the fiscal year — i.e., `publist92.dbf`. The `publist`*nn*`.dbf` file for the current fiscal year contains a list of `\<author code>`s and `\<subject code>`s for reference by the various people who are allowed to enter information into this file. The `\authorname` usually is the username of the author, and we classify publications into fourteen subject areas.

When a paper is published, `\submitted` is changed to `\published` and the `\volume{}`,`\page{}` and `\pubyear{}` information is added, and `\reprinttrue` is appended before the `\endref` to indicate we have received a published 'reprint' which corresponds to the reference information. If no reprint has been received, then `\reprintfalse` may be appended (this is also the default).

A list of `<journal code>`s and the journals they represent is maintained on 183 different journals. Some of the abbreviations were adopted from the Science Citation Index (1990) to ensure the uniqueness of our commands. Journal abbreviations that weren't found in this issue were created — there were quite a few in this category.

With the above "counters" we can generate reports by month, year, author, research area, and by individual journal, plus we can report how many have been published, how many were submitted, how many are 'to appear', how many technical reports (unpublished work), and published conference proceedings have been done by our researchers.

Lists. `allscrinames.list` is a list of all SCRI authors (i.e., those included in the annual report and proposals to date). This file notes whether each author is still here [`\localfalse` or `\localtrue` (which is the default)], and whether they are a member of the Lattice Gauge Theory Group [`\latticetrue` or `\latticefalse` (which is the default)]. Other helpful notations can be added as they arise. These conditions are noted above the author's name in the input file. Some examples would be:

```
\global\latticetrue
\name{Berg, Bernd A.}{berg.prep}
  (LGT Member, Local)
.
\global\localfalse
\name{Berger, Mordechai}{berger.prep}
 (Not LGT, Not Local)
.
\name{Burnette, Donna E.}{burnette.prep}
(Not LGT, Local)
```

The output file name is the same as the author code. Directing the output to the appropriate fiscal/super file is done during execution according to the user's interactive command.

`jnlabbrv.tex` and `emptyjnls.tex` contain the definitions necessary to create uniform output for each journal name. `jnlabbrv.tex` contains the actual meaning of the abbreviations.

`subjectabbrv.tex` contains the definitions necessary to create uniform output for each subject/discipline area.

Divider files. `publist.tex` is run interactively to separate the preprints by author or to generate a publications list since the beginning of SCRI, depending upon the option you enter at the beginning prompt. (See Appendix A for a definitive portion of the macro for this file, and Appendix B for an example of the interactive commands.) We simply write out to 13 files at a time, close them and then open another 13 files until the end of the run. At the termination of a run, you have either `\input` all of the database files and written out 178 author files, **or** you have `\input` 178 files and written out one file. Additional subsets can be created if the need arises and requires little effort (a change in the `\input` file) to produce the same information for a different period of time.

`subject.tex` does the actual division of files into the fourteen discipline areas.

`journals.tex` separates preprints by journal, regardless of what state they are in (published, accepted, submitted). `jnlspub.tex` separates only those preprints that have reached a published status. An entry identical to that used in `publist.tex` is required to accomplish the separation (substituting the journal code for the author code.

Report files. `monthly.tex` is used to run the various reports we do. An example of the interactive commands to generate reports is shown in Appendix C.

The author files all perform the same essential function, the only difference being in the usage of the conditions talked about earlier (`\localfalse` and `\latticetrue`). `authors-lattice.tex` extracts the preprints for the authors that are members of the LGT Group; `authors-local.tex` extracts the preprints for the authors that are still at SCRI; `authors-scriline.tex` extracts the preprints for the SCRI-funded faculty line authors; and `authors-superpub.tex` extracts the preprints

for everybody since the beginning of SCRI — it also includes the preprints by external authors.

`subject-superpub.tex` reports the preprints by subject area since SCRI began.

Running `journals-superpub.tex` creates a report based on the files generated by `journals.tex` (everything since the beginning of SCRI regardless of its status).

Designing a User Friendly Interactive Environment for Generating Reports

One of our goals was to make using this setup as simple as possible. We wanted this information to be public and easy to access for those who might take an interest. We also didn't want everyone at SCRI to have to be a TEXpert to be able to benefit from our development. To that end, we have created an interactive menu which reads input from the screen (see Appendices A and B), initializes another series of conditionals, and generates the requested report based on user input.

We have evidently been successful, for some of our scientists are now using their `authorname.super` files to update their own *vitae*, as well as finding preprints in various subject areas. We recently began posting our lattice field theory publications to an on-line mail server.

Summary

We began using TEX to generate reports first in 1986, one year after first being introduced TEX. At that time, our scientists used `plain` TEX, or submitted handwritten copy; LATEX was not used much at the time, or we might have tried to use BIBTEX. We have learned much about the programming capabilities of `plain` TEX over the years, and though the present program might be much more sophisticated, we have been satisfied with its simplicity. This simplicity has allowed us to easily modify the programs, often on short notice; and it brings to mind a quote from *The TEXbook* (page 373):

> ... Always remember, however, that there's usually a simpler and better way to do something than the first way that pops into your head. You may not have to resort to any subterfuge at all, since TEX is able to do lots of things in a straightforward way. Try for simple solutions first.

One of the ever-present problems which confronts those of us in "production" is that we are not often given time to learn more, or simply to "play", and unless we are lucky enough to have programming support we are confined to "getting the job done", rather than exploring avenues of getting the job done easier, or prettier, or faster. For some reason, our scientists almost universally wait until the last minute and then send us a file that has a deadline of "a week ago".

In all, our programs total several thousand lines and represent some seven report-generating files, which input five definition or macro files, to generate up to 13 reports. (No easy process to explain!)

It takes a little over two hours of cpu time to run `publist.tex`, approximately 40 minutes to run `authors-superpub.tex`, approximately five minutes to run `subject.tex`, and approximately two minutes to run the monthly report files. In the world of today's workstations, this overhead is negligible.

Bibliography

Institute for Scientific Information. "Lists of Source Publications." pages 66-85 (arranged by ISI abbreviations), *Science Citation Index*, vol. 1, Philadelphia, PA, 19104: University Science Center, 1990. (Also pages 86 – 106, arranged by full title.)

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

A. publist.tex Example

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\input publist.macros
\input emptydefs
\input emptyjnls
\input jnlabbrv
\input subjectabbrv
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\immediate\write16{}
\immediate\write16{IS A PRINTOUT OF THE ENTIRE PUBLICATIONS LIST REQUIRED (Y/N)?}
\immediate\write16{}
\message{--> }}
.
.
.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\immediate\write16{}
\immediate\write16{Enter S if you want the SUPERPUB files separated}
\immediate\write16{Enter P if you want the file for the current fiscal year separated}
\immediate\write16{}
\message{--> }}
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\global\read-1 to \datain
.
.
.
%
\if\runtype\s%
\def\fdir{preprints:[superpub]}%
% \def\fdir{[.test]}
\def\fextension{.super}
\fi%
\if\runtype\p%
\def\fdir{preprints:[fiscal]}%
% \def\fdir{[.test]}
\def\fextension{.fiscal}
\fi%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\if\printout\y{\global\def\endref{%
\hsize=7truein\parindent=0pt\baselineskip=12.4pt \parskip=6pt%
%
\if\currentptype\a\vtop{\line{\okbreak%
    \vtop{\hsize=2in\noindent\raggedright%
    \ifvoid\refnum{}%
    \else\bf FSU-SCRI-\unhbox\refyear\unhbox\refatype-%
\unhbox\refnum\break%
    \fi%
    \unhbox\refauthor\par\unhbox\refsubject}\hfill%
\hbox to .5truein{\hfill\unhbox\refdate\hfill}%
    \hfill\vtop{\parindent=0pt\hsize=4in\raggedright%
\ifvoid\reftitle{}%
\else\unhbox\reftitle\par%
\fi%
    \ifvoid\refjournal{%
\ifvoid\refbook{}%
\else In: \unhbox\refbook%
\fi%
```

```
\ifvoid\refeditor{}%
\else,\ \unhbox\refeditor, ed.%
\fi%
\ifvoid\refeditors{}%
\else,\ \unhbox\refeditors, eds.%
\fi%
\ifvoid\refvolume{}%
\else,\ \unhbox\refvolume%
\fi%
\ifvoid\refpage{}%
\else,\ \unhbox\refpage%
\fi%
\ifvoid\refpublisher{}%
\else,\ (\unhbox\refpublisher%
\fi%
\ifvoid\refpubyear{).}%
\else,\ \unhbox\refpubyear).%
\fi%
}%
    \else\unhbox\refstatus\unhbox\refjournal%
    \ifvoid\refvolume{}%
    \else,\ \unhbox\refvolume%
    \fi%
    \ifvoid\refpage{}%
    \else,\ \unhbox\refpage%
    \fi%
    \ifvoid\refpubyear{.}%
    \else\ (\unhbox\refpubyear).%
    \fi%
    \fi%
    \ifvoid\refextra{}%
    \else\ \unhbox\refextra.%
    \fi%
    }}%
\bigskip
}\fi%
%
\if\currentptype\p\vtop{\line{\okbreak%
.
\if\currentptype\s\vtop{\line{\okbreak%
.
\if\currentptype\t\vtop{\line{\okbreak%
.
\if\currentptype\z\vtop{\line{\okbreak%
.
.
.
}\fi%
\resetvars}%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
.
.
.
\newcount\filezero \filezero=0
\newcount\fileone \fileone=1
\newcount\filetwo \filetwo=2
\newcount\filethree \filethree=3
\newcount\filefour \filefour=4
\newcount\filefive \filefive=5
```

```
\newcount\filesix \filesix=6
\newcount\fileseven \fileseven=7
\newcount\fileeight \fileeight=8
\newcount\filenine \filenine=9
\newcount\fileten \fileten=10
\newcount\fileeleven \fileeleven=11
\newcount\filetwelve \filetwelve=12
\newcount\filethirteen   \filethirteen=13

.
.
.
```

The above 13 counters are used, emptied and reused until all publications have been written out to individual files.

B. Interactive Commands to Divide Files

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This preamble appears at the beginning and
% explain how to input data interactively.
{\obeyspaces\immediate\write16{}
\immediate\write16{}
\immediate\write16{}
\immediate\write16{                     FLORIDA STATE UNIVERSITY}
\immediate\write16{           SUPERCOMPUTER COMPUTATIONS RESEARCH INSTITUTE}
\immediate\write16{}
\immediate\write16{          SEPARATOR OF INDIVIDUAL SCRI AUTHOR'S PUBLICATIONS
\immediate\write16{                   (USING PUBLIST85.DBF -- present)}
\immediate\write16{}
\immediate\write16{IS A PRINTOUT OF THE ENTIRE PUBLICATIONS LIST REQUIRED (Y/N)
\immediate\write16{}
\message{--> }}
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\global\read-1 to \datain
%
\def\stripspace#1 \next{#1}
\def\stripzero0#1\next{#1}
\edef\datain{\expandafter\stripspace\datain\next}% strip \datain's space
%
\def\next#1\endname{\uppercase{\global\def\printout{#1}}}
%
\expandafter\next\datain\endname   %make\datain uppercase
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
{\obeyspaces\immediate\write16{}
\immediate\write16{Enter S if you want the SUPERPUB files separated}
\immediate\write16{Enter P if you want the file for the current fiscal year sep
\immediate\write16{}
\message{--> }}
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\global\read-1 to \datain
%
\def\stripspace#1 \next{#1}
\def\stripzero0#1\next{#1}
\edef\datain{\expandafter\stripspace\datain\next}% strip \datain's space
%
```

```
\def\next#1\endname{\uppercase{\global\def\runtype{#1}}}
%
\expandafter\next\datain\endname   %make\datain uppercase
%
\if\runtype\s%
        \def\fdir{preprints:[superpub]}%
%       \def\fdir{[.test]}
        \def\fextension{.super}
\fi%
\if\runtype\p%
        \def\fdir{preprints:[fiscal]}%
%       \def\fdir{[.test]}
        \def\fextension{.fiscal}
\fi%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

C. Interactive Commands to Generate Reports

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This preamble appears at the beginning and
% explain how to input data interactively.
{\obeyspaces\immediate\write16{}
\immediate\write16{}
\immediate\write16{}
\immediate\write16{                        FLORIDA STATE UNIVERSITY}
\immediate\write16{              SUPERCOMPUTER COMPUTATIONS RESEARCH INSTITUTE}
\immediate\write16{}
\immediate\write16{                 INTERACTIVE PUBLICATIONS LIST GENERATOR}
\immediate\write16{}
\immediate\write16{}
\immediate\write16{ENTER "Q" FOR A MONTHLY REPORT}
\immediate\write16{ENTER "R" FOR A COMPLETE PUBLICATIONS LIST}
\immediate\write16{ENTER "Y" FOR A COMPLETE LISTING OF ABSTRACTS}
\immediate\write16{}
\immediate\write16{ENTER "A" FOR A LIST OF PUBLICATIONS WHICH ARE "TO APPEAR"}
\immediate\write16{ENTER "C" FOR A LIST OF CONFERENCE PROCEEDINGS}
\immediate\write16{ENTER "E" FOR A LIST OF EXTERNAL PUBLICATIONS}
\immediate\write16{ENTER "M" FOR A LIST OF MISCELLANEOUS PUBLICATIONS}
\immediate\write16{ENTER "P" FOR A LIST OF PUBLISHED PUBLICATIONS}
\immediate\write16{ENTER "S" FOR A LIST OF SUBMITTED PUBLICATIONS}
\immediate\write16{ENTER "T" FOR A LIST OF TECHNICAL REPORTS}
\immediate\write16{ENTER "V" FOR A LIST OF VISITORS PUBLICATIONS}
\immediate\write16{}
\message{--> }}
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

An Audio View of (LA)TEX Documents

T.V. Raman
Center for Applied Mathematics
Cornell University
Ithaca NY 14853-6201
Phone: (607)255-7421
Internet: raman@cs.cornell.edu

Abstract

Till now, (LA)TEX has been used to generate typeset documents. This paper points out an alternative view — the generation of audio renderings. The (LA)TEX typesetting source captures a lot of document structure which is then used to typeset the document. This can be exploited to convey document structure in audio renderings of the document as well. This becomes especially important when dealing with mathematical documents. Our approach attempts to develop notions of audio formatting similar to the well-understood notions of visual formatting and layout. Our goal is to do for the audio document what (LA)TEX has done for the printed document.

Introduction

Lack of ready access to current mathematical and technical literature has been a major stumbling block throughout my career. I first became interested in trying to use (LA)TEX to overcome this problem when I received the LATEX source for the lecture notes in a course on the Design and Analysis of Algorithms at Cornell. I use a talking computer, since I am visually handicapped, and availability of technical documents on-line meant that I could have them read by my computer. However, I soon realised the futility of having the computer read out (LA)TEX source directly. The only other effective solution available at the time, namely to have the printed text read aloud, was clearly inadequate. Of course, I had the option of trying to get the printed output read out using a reading machine, but in the case of texts with heavy mathematical content, this continues to remain impracticable at present. Current OCR (OPTICAL CHARACTER RECOGNITION technology is incapable of handling typeset mathematics.

Using the (LA)TEX sources as a starting point for generating audio renderings was a temptation too hard to resist. (LA)TEX captures a lot of syntactic and sometimes even semantic information about the document content, and this information can be used for more than just typesetting the document. It could be used equally well in producing high quality computer-generated audio renderings of the document. This is especially true when it comes to reading mathematical texts. Complicated mathematical constructs, which prove a major stumbling block for conventional OCR technology, present far less of a problem, since the (LA)TEX constructs that are used to produce the final output capture a lot of information about what is being laid out on paper. Thus, this information can be effectively captured at the (LA)TEX source level and exploited in producing audio. I therefore started work on a system for generating audio renderings of technical documents presented in the form of (LA)TEX mark-up source.

A First Attempt

I first worked on a program that would transform (LA)TEX source to a form more suitable to be read out by a talking computer. Reading the (LA)TEX source directly is impractical, since you have to listen to a stream of "backslash" and other extraneous utterances.

This first attempt resulted in the development of TEXTALK (Raman, 1991), a program that I continue to use for my day-to-day work as a graduate student at Cornell. This program carries out simple transformations on the (LA)TEX source and the resulting text can be viewed using standard UNIX tools. The text which is displayed on the screen can then be effectively read out by the talking computer. Thus presented with the expression:

$$\frac{1+\sqrt{5}}{2}$$

the program transforms the above text to:

The fraction with numerator 1 + square root of 5 and denominator 2 end of fraction

This is much more intelligible than the following cryptic utterance which would be generated if the (LA)TEX source were being directly read:

> *dollar dollar backslash frac left brace one plus backslash sqrt left brace five right brace right brace left brace two right brace dollar dollar*

TEXTALK modified (LA)TEX mark-up source corresponding to mathematical notation and generated text that would parallel what a human reading out the printed result would say. See the Appendix for a more detailed example. I obtained immediate direct access to several text books whose authors kindly agreed to make available the on-line sources. Currently I also have access to the on-line sources for the AMS bulletins. Thus, the program makes the latest mathematical publications accessible to me.

This first attempt uncovered a lot of interesting issues, which then led naturally to the next step, namely, developing notions of audio formatting analogous to the well-understood notions of visual formatting. Though the program as it works is eminently usable, the audio renderings generated are not as effective in conveying the complete structure of the document. Sub-expressions occurring in a complicated expression make perfect sense when handled by the system, but it is still difficult to comprehend extremely complicated mathematical expressions, especially in terms of understanding how the various sub-expressions interact with one another. The example in the Appendix, which shows the text generated for a complex math expression, makes these shortcomings explicit.

A Rigorous Approach

The initial implementation revealed the need for developing a rigorous approach to audio formatting. It also became apparent that in order to do full justice to the audio, carrying out transformations based on a linear scan of the (LA)TEX source itself was not adequate. Even though (LA)TEX source contains a lot of useful information about document structure, the earlier approach of string substitution, i.e., scanning the source and applying simple transformations fails to fully exploit all of this information. The transformations carried out tended to be local in nature as string substitution ignores global structure and this was identified as a principal cause of the ineffectiveness in conveying global structure. The above becomes clear when we compare the readings corresponding to a complicated expression generated by a trained and experienced reader with those generated by the system. See examples 2 and 3 in the appendix for a comparison. In fact even a straightforward transcription of the text as present in the recordings from the Recordings for the Blind (RFB) loses a lot of information. This is because the trained reader inserts appropriate pauses and uses other prosodic cues to convey the nesting of complicated sub-expressions. This shows clearly that a system that attempts to read out the text resulting from carrying out simple transformations to the (LA)TEX source will be unable to convey such structural information. The trained RFB reader is able to insert appropriate cues into the spoken text only after having parsed and re-parsed the expression. This shows a clear need for first constructing higher level representations of the expression in order to improve the quality of the audio rendering. Thus, there are two steps to audio rendering:

- generate the text to be spoken, and
- "audio format" this text, i.e., insert appropriate cues into the spoken text in order to convey structure.

A two-step approach. The preceding discussion shows that a better approach is to subdivide the problem into:

1. building a high-level model of the document by parsing the (LA)TEX source, and
2. generating audio renderings by applying appropriate audio formatting rules to the resulting structure.

Advantages. This approach no longer suffers from the drawbacks alluded to earlier. This is because the audio renderings that are now driven off a high-level representation of the document can draw upon global knowledge of document structure. Thus, while sub-expressions continue to be formatted for audio by applying transformations as before, the audio renderings can now capture information about how various entities appearing in the document relate to one another. The human reader conveys such structural information using prosodic cues. When using computer-generated speech, the range of prosodic cues that we can use is limited in comparison. However, these can be augmented by using non-speech audio cues. By *non-speech* audio cues, we mean short chords of music, beeps etc. that can be used to convey non-textual content. These can prove extremely useful, since they can be used to cue the listener to extra-textual content without introducing unnecessary verbiage in the audio renderings. In addition, the audio documents generated from such high-level document structures will be capable of allowing the user to browse the document.

Generating High-Level Document Models from (LaTeX) TeX Source

This section details the approach we have taken to solving the first of the two sub-problems outlined in the previous section. Well-written (LaTeX) TeX documents capture document structure using macros designed to reflect logical rather than layout structure. This fact is heavily relied upon when generating high-level models, given specific instances of (LaTeX) TeX documents. This forces certain constraints on the type of (LaTeX) TeX documents that such a system will be able to handle. These constraints are described in the following subsections. We then give an approach for generating such high-level models for (LaTeX) TeX documents satisfying these constraints.

Defining high-level models. We think of high-level models of the documents as being given by abstract syntax trees corresponding to a specific language $\mathcal{L}$. Different document types satisfy different languages, and thus have different high-level representations. The abstract models we define are thus specific to a given class of documents. For the present we will consider the `article` style of LaTeX.

A hierarchical structure. Documents conforming to the `article` style of LaTeX have a clear hierarchical structure. The document divides neatly into a simple tree structure where the subtrees correspond to various structural units such as section, subsection, etc. LaTeX documents have this structure clearly tagged by the use of standard LaTeX constructs and this allows us to get at the high-level structure (Lamport, 1986). Similar structures are also easy to obtain from well-written TeX documents where the structure is marked up using macros from standard packages such as `plain` TeX (Knuth, 1984). However, since TeX does not always insist on the use of predefined structures for marking up the document, this step can prove difficult when dealing with raw TeX documents.

Defining a recognizable class of (LaTeX) TeX documents. TeX as described by Knuth (1984, 1986) is a powerful typesetting language and embodies many features of a programming language. By providing primitives normally found in a programming language it affords immense flexibility to the designer of a document. However, with this flexibility come a lot of problems, since such power in the hands of an average user can prove dangerous.

> All power corrupts and absolute power corrupts absolutely!

This statement is true in the world of TeX documents as well. A properly prepared TeX document uses the power of the language sparingly and avoids mixing typesetting commands with document content. Using well-designed formats results in (LaTeX) TeX source that clearly reflects the document structure. However, the ease with which new constructs can be defined in TeX means that the above principles are often violated.

Documents which rely on absolute commands like `\vskip` to achieve document structure by providing the right visual effect present serious problems to a program that is trying to build a higher level model of the document. This is to be expected since the use of such absolute commands within the text of an electronic document indicates an assumption that the electronic source will be used only for typesetting the document. Thus, the first and most important constraint that we impose on the class of (LaTeX) TeX documents that we handle is that document structure be clearly tagged using only standard macro packages. Thus our current parser recognizes an enumerated list in LaTeX which is clearly marked up, i.e., each new item is explicitly tagged using `\item`. However, if an author chooses to mark-up certain items in an enumerated list by using `\item` and other items by `\foo`, where `\foo` has been defined by the author to generate the right visual effects required to signal a new item, the parser fails to recognize some of the items in the list.

Classifying (LaTeX) TeX macros. The level of complexity present in constructing an abstract model of a document given its (LaTeX) TeX source is directly determined by the type of macros used by the author. Macros can be used to mark up document structure, for laying out specific structures and objects, and for achieving visual effects. As pointed out above, macros are also used to augment predefined formats. In addition, macros are also widely used to make the task of keyboarding easier. This multiple use of macros causes some difficulty when constructing high-level models of a (LaTeX) TeX document. In order to define the class of (LaTeX) TeX documents we can handle, we need to classify TeX macros according to how they are used. This will allow us to clearly define the class of macros that we can handle in our document recognition step and will, in effect, define the class of documents that the system will be able to recognize. Further, this classification step will also indicate to what extent we should expand macros during the recognition step and how the abstract model is to capture macro calls.

In the following, $\mathcal{M}_U$ is used to denote the universal set of all macros. The various subsets are denoted by appropriate suffixes.

Graphic. This subset will be denoted by $\mathcal{M}_G$. Macros that provide primitive typesetting operations such as the `\kern` and `\hrule` control sequences in TEX are typical examples of such macros. They are characterized by their visual nature. Explicit use of such macros in a document does not provide sufficient information to allow for the construction of an abstract model.

Graphic macros representing standard objects. This set of macros will be denoted by $\mathcal{M}_O$.

The term *standard object* is used here to refer to commonly occurring entities such as integrals and fractions. These objects have a standard visual template according to which they are typeset, and a properly composed document renders these visually by using specialized macros that take appropriate arguments. Thus, by their very nature, such macros capture a functional representation of the object. When confronted by such macros in a document, it is unwise to try and expand them any further in terms of the lower-level control sequences from which they have been constructed. Thus, the `\frac` macro of LATEX contains all the higher-level information we can get about the object it renders, and the model we build should capture this call.

Another difficult subset of macros that belong to this category are special symbols built up with primitive control sequences. This is typical of complicated combinations of primitive macro calls that are used to create special visual effects. When building the higher-level model of the document, we need to capture the essence of what is being conveyed by the use of such macros, rather than the result of expanding the macro call. Thus if `\Real` has been built up using a set of primitive *visual macros* to produce a real number symbol, the abstract model should stop by capturing just the macro call `\Real` rather than attempting to expand it any further. In fact, expanding the call will actually eliminate information.

In an ideal world the principal type of macros one would encounter when parsing the typesetting source would be elements of $\mathcal{M}_O$. However, life is not so simple, and often electronic documents abound with the use of lower level control sequences. Further, $\mathcal{M}_G \cap \mathcal{M}_O = \emptyset$ does not always hold. It can be argued that the `\Real` control sequence discussed earlier actually belongs to $\mathcal{M}_G$. This would be true if we did not know what the author intended to represent by the use of the macro, which could often be the case.

The above is also true of the use of TEX control sequences such as `\atop` and `\over`, which often require some knowledge of the context in which they are used in order to come up with a semantic interpretation of their use. Thus, the use of `\atop` in a nested subscript often means conjunction, while it means something entirely different when used in rendering a Legendre symbol. Consider the following example (Knuth, 1984:145, 320 (**ex. 17.9**)):

$$\sum_{\substack{1\le i\le p\\ 1\le j\le q\\ 1\le k\le r}} a_{ij} b_{jk} c_{ki}$$

which is produced by:

```
$$\sum_{{\scriptstyle 1\le i\le p
\atop\scriptstyle 1\le j\le q}
\atop\scriptstyle 1\le k\le r}
a_{ij} b_{jk} c_{ki}$$
```

Macros for simple text substitution. This class of macros will be denoted by $\mathcal{M}_T$. These macros are typically used to make the task of keyboarding easier. In most situations it is safe to expand these macros since their expansion does not lead to loss of structural information. However, once again $\mathcal{M}_O \cap \mathcal{M}_T = \emptyset$ is not always true.

Consider the use of the macro `\reader` in the following sentence.

"We are working on a `\reader` for electronic documents."

where `\reader` has been defined elsewhere as

```
\def\reader{new exciting reading machine}
```

This macro clearly belongs to $\mathcal{M}_T$ in the context in which it is described. Expanding it directly will lead to the text "We are working on a new exciting reading machine for electronic documents." This can now be easily rendered in audio.

However, viewed from a different perspective, i.e., the use of `\reader` as a logo, this macro could well be said to be in $\mathcal{M}_O$. In the context of audio formatting, we may wish to render the result of the call to the `\reader` macro differently.

Recognizing the type of a macro. Recognizing the class to which a given macro belongs is a hard problem. In general, no universal classification will always hold, as is clear from the previous description. However, we can use some simple heuristics to classify a major subset of the commonly occurring macros. Clearly all the primitive typesetting operations provided by TEX in terms of putting marks on paper belong to $\mathcal{M}_G$. Further macros that are known not to be in $\mathcal{M}_G$ but take arguments are typically in $\mathcal{M}_O$, since the use of macro arguments normally indicates that a template is being filled up. Macros that which do not fall into either category

now fall into $\mathcal{M}_T$. Given more information about the context in which the macro is being used, it becomes possible to further refine the above classification.

Summary of constraints. To summarize, here are the constraints we need to impose on the class of (LaTeX) TEX documents we can handle:

1. document structure should be clearly marked up,
2. explicit use of absolute commands should be avoided, and
3. use of macros should reflect semantics and logical structure rather than physical layout.

These constraints have been introduced while discussing the global document structure. However, they hold equally well when considering specific components of a document, such as mathematical expressions occurring within the document.

Format-independent techniques of information capture. The above discussion also reveals the need for developing a framework for representing information in electronic documents independent of any single "display" method. TEX goes a long way in achieving this for mathematical documents. However, since all the primitive operators used to achieve this are also available to the average user, TEX documents do not always conform to the constraint that information in a document be represented independent of formatting details. As these concepts become better understood, we need to work towards the development of a language for representing structured information in electronic documents. Some of this has already been achieved by SGML (Standard Generalized Mark-up Language, ISO, 1990) and there is a clear need to carry over this work to cover mathematical documents as well. Development of such format-independent techniques of information capture will allow us to provide alternative methods of accessing the same information structures.

Reading Mathematics Aloud

The previous section pointed out the need for high-level information capture independent of specific formatting techniques. Given high-level information structures, we need to develop adequate techniques for accessing this information using alternative perceptual modalities such as audio. This section addresses the various questions that arise in developing an effective notational system for mathematics in the audio world. In order to do this, we first analyze how visual notation works and attempt to apply some of what we learn to the audio world.

Features of visual math notation. Traditional math notation fully exploits the two-dimensional nature of the visual tablet. It is therefore not linear but achieves conciseness by using subscripts and superscripts. It relies on the eye's ability to move quickly across the paper, and uses visual cues such as delimiters of different sizes to cue these structured movements. Written mathematics is not linear in two different senses of the term, i.e., space and time.

1. It uses a two-dimensional display.
2. It is not linear in time since the eye is able to move back and forth across the paper seemingly at will.

Both of these features are absent in traditional spoken mathematics. Spoken mathematics on tape is linear in time. In addition, it tends to be wordy since there is no standard way of alerting the listener to complicated syntactic constructions other than describing these verbosely.[1] This can be directly attributed to the apparent lack in audio of the two-dimensional nature of the printed medium. This lack of two-dimensionality is overcome by the reader inserting words such as "open quantity" and "raised to the quantity" in order to cue the listener to the presence of complex constructions. Use of such cues, though effective, causes the audio rendering to be necessarily verbose, making it difficult to grasp the essence of what is being conveyed. Mathematical notation relies on conciseness to express high-level concepts, and as the complexity of the expressions being handled increases, the resulting verbiage in the spoken equivalent of the written expression renders it practically unusable. We need adequate audio substitutes for these features of mathematical notation if we are to have any hope of effectively conveying mathematics using audio.

Spoken mathematics. The previous paragraph takes care to speak of *mathematics* in *audio* rather than merely referring to "spoken mathematics". This choice of terminology is intentional. One of the ways traditional spoken mathematics conveys complex inter-relationships between sub-expressions is to use prosodic cues within the speech (see the appendix, example 2, for an explicit example). In order to achieve the full expressiveness of a human reading mathematics we need to use a lot of prosodic cues in the speech. Though computers of today can talk intelligibly they are still a long way from achieving

[1] See Chang (1983), which is used as a guideline by RFB for reading mathematical texts.

this degree of expressiveness. We therefore need to augment the computer's speech ability by a well-designed system of audio cues which can then be used to better convey extra-textual content when reading complicated expressions.

Multiple channels of audio. Non-speech audio cues can be synchronized with the speech output. Thus, by using multiple channels of audio output, i.e., by having both speech and non-speech audio cues playing at the same time, and exploiting directionality of sound, we can offset the disadvantages resulting from not having a two-dimensional display. In fact, the audio document is not restricted to a flat display, and proper use of audio cues can result in effective communication of complex constructs.

Browsing in an audio document. The previous subsection addresses the problems resulting from traditional audio documents being linear in space. This subsection in turn addresses the problems resulting from the fact that conventional recordings on tape have been linear in time.

A serious difficulty faced when listening to the recording of a complex expression is that the listener is forced to retain the entire expression. Thus, comprehending spoken mathematics demands a longer attention span. In fact, often it becomes impossible to remember the beginning of a complex expression by the time one has reached its end. This is clearly evinced by the reading of Faa de Bruno's formula presented in the Appendix of this paper. Visual notation, by using different sized delimiters, different levels of subscripts, etc., cues these structured movements, and thereby allows the eye to move around the printed expression. Analogously, we need to allow the listener to move around the expression and access parts of it at will. This will obviate the need for the listener to retain the entire expression in memory.

These structured movements can be performed using the high-level model for the expression that has been constructed at the recognition step. This is one of the major advantages of first recognizing the structure before generating audio renderings.

Conclusion

This paper describes an audio view of (LA)TEX documents. Electronic typesetting source can be used to generate audio documents as well. In order to do this effectively, we need to recognize document structure from the electronic source. Audio renderings will then be driven from this high-level structure. There is a need to develop notions of audio formatting analogous to the well-understood notions of visual formatting. Finally, we need to better understand how visual browsing works in order to build into the system the ability to provide the same functionality in the audio setting.

Acknowledgements

I would like to thank Prof. David Gries for his help and advice in my work. I would also like to acknowledge Xerox Corporation for supporting this work both in the form of summer support during 1991 and 1992 and an equipment grant that enabled the project to acquire a MultiVoice speech synthesizer. I would also like to thank Prof. Bruce Donald for his advice and help and the Cornell Computer Science Robotics and Vision Laboratory for their support during the last year.

Bibliography

Chang, Larry A. *Handbook for Spoken Mathematics*. Livermore, CA: Lawrence Livermore National Laboratory, 1983.

Knuth, Donald E. *The Art of Computer Programming*, Vol. 1, 2nd ed. Reading, Mass.: Addison-Wesley, 1973.

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

Knuth, Donald E. *TEX: The Program*. Reading, Mass.: Addison-Wesley, 1986.

Lamport, Leslie. *LATEX: A Document Preparation System*. Reading, Mass.: Addison-Wesley, 1986.

International Standards Organization. *Information Technology — SGML Support Facilities — Techniques for Using SGML*. Draft, 1990.

Raman, T.V. TEXTALK. *TUGboat* 12(1), page 178, 1991.

Information Processing — Text and Office Systems — Standard Generalized Markup Language (SGML). October 1986. ISO 8879–1986 E.

Appendix: An example of complex math expressions

This appendix gives the TEXsource for a complicated mathematical expression, the transformed text generated by TEXTALK, and finally the text as read on the RFB recording of the same expression.

Note: This piece of mathematical text is taken from Knuth (1973:50 (**ex. 21**)):

1. The TEXsource

```
{\bf 21.} {\em [HM25]} (Faa di Bruno's formula.)
Let $D^k_x u$ represent the $k$th derivative of a function $u$ with
respect to $x$.  The ``chain rule'' states that $D^1_xw = D^1_u w
D^1_x u$.  If we apply this to second derivatives, we find $D^2_xw =
D^2_u w (D^1_x u)^2+D^1_u w D^2_x u$.
Show that the {\em general formula\/} is
$$D^n_xw =
\sum_{0\le j\le n}
\sum_{\scriptstyle k_1+k_2+\cdots+k_n=j
\atop {\scriptstyle k_1+2k_2+\cdots+nk_n=n
\atop  {\scriptstyle k_1,k_2,\ldots,k_n\ge0 }}}
D^j_u w \frac{n!}{k_1!{(1!)}^{k_1} \cdots k_n!{(n!)}^{k_n}}
{(D^1_x u)}^{k_1} \cdots {(D^n_x u)}^{k_n}.$$
```

The above piece of TEX code was keyed in while listening to the reading of the expression on the RFB tape. This shows up an interesting fact about ordering of subscripts and superscripts. In this case the reader has read the superscript first, and the TEX source reflects this in that it uses `D^1_x` rather than the more standard `D_x^1` as recommended in *The TEXbook*. At this time, there seems to be no reason to choose the reader's order of speaking the subscript after the superscript in the general case as against the order one would use if scanning the standard TEX usage linearly. In the case of Faa de Bruno's formula, the reader has used his interpretation of D^1_x in making this choice. In the interest of being able to easily parse the TEX source, we need to stick to either one of the two orderings when writing TEX documents. The order in which the subscripts and superscripts are eventually read can then be decided at a later stage in the audio formatting.

2. Text taken verbatim from RFB's recording

Exercise 21 has a rating of cap h cap m 25
Faa de Bruno's formula.
Let D super k sub x of u represent the kth derivative of a function u with respect to x. The chain rule states that cap d super 1 sub x of w equals cap d super 1 sub u of w cap d super 1 sub x of u. If we apply this to second derivatives, we find that cap d super 2 sub x of w equals cap d super 2 sub u of w times the quantity cap d super 1 sub x of u quantity squared plus cap d super 1 sub u of w times cap d super 2 sub x of u. Show that the general formula is cap d super n sub x of w equals the summation from 0 less than or equal j less than or equal n of the summation over k sub 1 plus k sub 2 plus and so forth plus k sub n equals j, k sub 1 plus 2 k sub 2 plus and so forth plus n k sub n equals n, k sub 1 comma k sub 2 comma and so forth comma k sub n greater than or equal to zero The quantity being summed is
cap d super j sub u of w times the fraction n factorial over k sub one factorial times one factorial raised to the k sub 1 times and so forth times k sub n factorial times n factorial raised to the k sub n the entire fraction times
the quantity cap d super 1 sub x of u closed quantity raised to the k sub 1 times and so forth times the quantity cap d super n sub x of u closed quantity raised to the k sub n

3. Transformed text generated by TEXTALK

21. [HM25] (Faa de Bruno's formula.)
Let D super k sub x u represent the k th derivative of a function u with respect to x . The "chain rule" states that D super 1 sub x w = D super 1 sub u w D super 1 sub x u . If we apply this to second derivatives, we find D super 2 sub x w = D super 2 sub u w (D super 1 sub x u) super 2 + D super 1 sub u w D super 2 sub x u . Show that the general formula is
D super n sub x w = sum sub 0 less than or equals j less than or equals n sum sub k sub 1 + k sub 2 + ellipses + k sub n = j and below that k sub 1 + 2 k sub 2 + ellipses + n k sub n = n and below that k sub 1

, k sub 2 , and so on , k sub n greater than or equals 0 D super j sub u w fraction with numerator n factorial and denominator k sub 1 factorial (1 factorial) super k sub 1 ellipses k sub n factorial (n factorial) super k sub n end of fraction (D super 1 sub x u) super k sub 1 ellipses (D super n sub x u) super k sub n .

4. The resulting formatted output

21. *[HM25]* (Faa de Bruno's formula.)

Let $D_x^k u$ represent the kth derivative of a function u with respect to x. The "chain rule" states that $D_x^1 w = D_u^1 w D_x^1 u$. If we apply this to second derivatives, we find $D_x^2 w = D_u^2 w (D_x^1 u)^2 + D_u^1 w D_x^2 u$. Show that the *general formula* is

$$D_x^n w = \sum_{0 \le j \le n} \sum_{\substack{k_1+k_2+\cdots+k_n=j \\ k_1+2k_2+\cdots+nk_n=n \\ k_1,k_2,\ldots,k_n \ge 0}} D_u^j w \frac{n!}{k_1!(1!)^{k_1} \cdots k_n!(n!)^{k_n}} (D_x^1 u)^{k_1} \cdots (D_x^n u)^{k_n}.$$

Model-Based Conversions of LaTeX Documents

Dennis S. Arnon
Xerox PARC
3333 Coyote Hill Road
Palo Alto, CA 94304 USA
415-812-4425; FAX: 415-812-4241
Internet: arnon@parc.xerox.com

Isabelle Attali
INRIA Sophia Antipolis
Route des Lucioles
06565 Valbonne Cedex, France
Internet: Isabelle.Attali@sophia.inria.fr

Paul Franchi-Zannettacci
University of Nice
CERISI
Sophia Antipolis
06561 Valbonne Cedex, France
Internet: pfz@essi.cerisi.fr

Abstract

We are creating a document conversion system based on modelling the logical structures of two broad categories of document types: *human-oriented* and *machine-oriented*. Each human-oriented, or *user*, type is a genre of documents that is well defined and well known to human authors and readers; for example, "scientific articles", "mathematical formulae" (e.g., $\frac{x^3+x^2-1}{x^2+1}$), or "limericks". Each machine-oriented, or *agent* type, is the set of legal data objects of a document processing tool, for example, "LaTeX documents" or "troff/EQN documents". Our models specify the abstract, rather than the concrete (i.e., surface) syntax, of particular documents. Thus we work with documents as tree data structures, whose concrete presentations have a surface syntax of no intrinsic interest. By so proceeding we hope to best utilize the computational sciences, and sidestep what we believe to be the red herring of "markup language".

Introduction

In contemporary document management systems, documents are often characterized chiefly by the format in which they happen to be represented at some instant of time. A *format* is simply the document encoding convention of a particular document processing agent; we thus prefer to refer to formats as *agent* document types. Formats are often classified into a rough hierarchy of "levels": image formats (TIFF, SunRaster, Fax, etc.) are said to be the lowest level, next are page description level formats (PostScript, InterPress, etc.), and finally are high level, or structured, formats (TeX , SGML DTD (Goldfarb, 1990), ODA DAP (Rosenberg, 1991), etc.).

For some classes of people, such as the implementor of such a document management system, the format-centric approach to document taxonomy is reasonable. These are people who in all likelihood have no contact with either the creators or the consumers of the documents, and their role is to be a "middleperson", or broker, between a creator whose output is in one format, and a consumer who wants the document presented in a different format. The document broker probably neither knows nor cares what the document is about, how it is organized, or

what its actual content is; instead, accepting a charter to render instances of one format in the other, as "faithfully" as possible.

Document brokering is not an appropriate view of another large class of document transactions, however. A collection of individuals who use different document processing tools may be co-authoring a single document; or, an author and a sophisticated reader; e.g., a colleague, may wish to share a document. In these instances, the "sender" and the "recipient" may have a good deal of common knowledge about a document's organization and content. Indeed, for optimal communication among themselves, the individuals in such groups want their respective document processing tools to fade into the background. They want instead to focus clearly on the "abstract" document that is the object of their interaction; i.e., an abstract intellectual entity that is independent of the particular software tools used to process concretizations of it. Furthermore, they undoubtedly have a common model of this abstract document (which we would say belongs to a *user* document type) in their respective mind's eyes. To convert the document from one format to another in these situations, it may be advantageous to know what the model is, and use it to "direct" or "govern" the format conversions. In these model-centric contexts, there are likely to be aspects of one format that do not map well to another format, in general. However, there will most likely be *some* way of encoding the relevant features of the abstract document in the destination format. By having the underlying abstract model in hand, and only in this way, we can maintain "knowledge" of which model features are encoded which way in which format, and thereby accomplish higher-quality format conversions for a community of model-centric document users.

In this paper we focus on a single user document type, that of "technical articles" (which may include mathematical formulae). We give a general model-building methodology, then specialize it to create models for the Article user type, and for agent types. Currently our system deals with two agent types, LaTeX documents and Tioga documents. (Tioga is a WYSIWYG editor for structured documents in the Cedar programming environment at Xerox PARC; see Swinehart, et al., 1986). We have created a software tool that supports conversions in which we are given a LaTeX or Tioga incarnation of a document asserted to be a technical article, and we want to convert it to the other agent type. We utilize distinct models for each user and agent type, and precisely specify the rules of interconversion for each $(agentType, userType)$ pair. An $agentType \rightarrow agentType$ conversion is then carried out by $agentType \rightarrow userType \rightarrow agentType$ mappings. Thus our tool converts a document from one format to another, while explicitly seeking to preserve its "technical-article-ness" as much as possible. This is what we mean by a *model-based conversion* of a document: a document believed to be a valid instance of some user type, and presented as an instance of some agent type, is converted to an instance of another agent type, in a manner as faithful to its user type as possible. The choice of particular agent types is secondary; the crux of our methodology is the "direction" of document format conversions by user type models.

The following section presents our general model-building methodology, based on the notion of a *formalism*, and our model for the Article user type. Our model-based approach to format conversion is divided into two steps: *analysis*; i.e., creating a valid user type model for the input document, and *synthesis*; i.e., rendering the model instance in the desired output format. We also call these steps *parsing* and *unparsing*. In subsequent sections, we discuss the parsing and unparsing of Tioga documents, and the parsing and unparsing of LaTeX documents. We concentrate on parsing Tioga documents to Articles and unparsing Articles to LaTeX , rather than the opposite direction. This reflects the fact that at the moment the least developed link in our system is our LaTeX to Article parser, a situation we are in the process of rectifying.

Currently our system is implemented in the Cedar programming environment (Swinehart et al., 1986), which in turn runs on top of UNIX and X Windows. Although we do not currently make actual use of the Centaur system (Borras et al., 1988), we are heavily influenced by the architecture and concepts of Centaur, and of attribute grammars (e.g., see P. Franchi-Zannettacci & D. Arnon, 1989, and A. Brown, H. Blair, 1990).

Modelling Articles with Formalisms

Terms. The first key component of our system is a single, universal data structure and surface syntax for labelled, n-ary, attributed trees. This we accomplish with a general tree manipulation package we have written, called Scrimshaw. "First order terms", "abstract syntax trees", "functional forms", and "expression trees" are additional equivalent names for the basic entities of Scrimshaw; we refer to all of them simply as *terms*. Atomic terms are identifiers, 32-bit integers, "reals" (i.e.,

IEEE standard floating point numbers), and character strings. Composite terms are "expression trees", whose "operator names" are identifiers, and whose "arguments" are (recursively) terms.

We implement document processing operations (e.g., parsing, unparsing, conversion) as *term rewriting* or *tree transformation* operations on Scrimshaw terms. The particular "function", from terms to terms, that any such operation represents, is not a part of Scrimshaw itself. Rather, each document type gives certain "interpretations", i.e., meanings, to certain terms of interest, and its operations map them to certain other terms which also have intended interpretations. All document types can equally make use of the general term data structures and operations that Scrimshaw provides, for what it considers to be uninterpreted terms.

Here is a portion of a Scrimshaw term representation of the Tioga form of this paper, which we hope will suffice to illustrate the nature of Scrimshaw Terms, in lieu of a full definition:

```
internalNode[
 nodeList[
  leafNode[
    format[title],
     contents[
       runList[
        text["Model-Based Conversions
              of LaTeX Documents"]
               ]
              ]
             ],
  leafNode[
    format[authors],
     contents[
      runList[
       text["Dennis S. Arnon ..."]
             ]
            ]
         ],
  leafNode[
    format[abstract],
     contents[
      runList[
       text[
        "Abstract: We are creating a
         document conversion system
         ... formulae\'\' (e.g., "],
       text["X",
        prop[
         $MathNotation,
         "quotient[diff[sum[power[name[x],
          number[3]], power[name[x],
          number[2]]], number[1]],
          sum[power[name[x], number[2]],
          number[1]]]"
           ] ],
      text[
        "), or ``limericks\'\'.
       Each machine-oriented ...   ."]
         ]
        ]
      ]
    ]
  ]
```

Formalisms: abstract vs. concrete syntax. A document *type* is a certain family of labelled, n-ary trees, or in other words, some subfamily of Scrimshaw terms. There is some finite set of label names (each of which is a Scrimshaw identifier) for nodes; a node's label is called its "operator". The set of label names is non-disjointly partitioned into subsets called *phyla*; thus, each operator belongs to one or more phyla. In addition, each leaf node has a *value* that is either an identifier, an integer, a character, a string, a term of some other formalism, or empty. For each operator, we specify the number (*arity*) and phyla (*types*) of its descendants. An operator is either *atomic*; i.e., it is a leaf node with zero descendants and a value, or a *fixed arity* operator, with some fixed number $n \geq 1$ of descendants, whose root operators belong respectively to phyla $P_1, ..., P_n$, or a *list* operator, capable of having zero or more descendants whose root operators all belong to a single phylum P. Atomic operators in one formalism, whose values are terms of another formalism, are our means of providing for the nesting of document types one within another, e.g., mathematical formulae within technical articles. The terms of a document type may be attributed, and the attribute (i.e., property) values may be crucial to the specification of the type model; in the limited space of this paper, however, we have little to say about attributes.

A family of terms specified as above is called a *formalism*, and its constituent terms are called the *abstract syntax trees* of the formalism. We implement each abstract type of documents (e.g., Article, math formula) as a formalism. Thus the "canonical" abstract representation of a document in our methodology is as an abstract syntax tree of some formalism.

Let us now illustrate these concepts with excerpts from the Article formalism we use to model technical articles. Owing to the limited space in this

paper these excerpts will stand in lieu of a full definition. Here are the specifications of the `article` and `header` operators in the Article formalism:

```
article → HEADER BODY END ;
header → TITLE AUTHORS ABSTRACT KEYWORDS;
```

is a 4-ary operator; the first descendant of a header must be of phylum `TITLE`, its second of phylum `AUTHOR`, etc. These phyla happen to contain only one operator each:

```
TITLE ::= title ;
AUTHORS ::= authors ;
ABSTRACT ::= abstract ;
KEYWORDS ::= keywords ;
```

so we are effectively requiring a single, 4-part structure in the header of an article. (We write operators in lower case and phyla in upper case throughout this section).

Our model of "text" in Articles is that it is a "list of text items". Thus we have a list operator "paragraph" defined by:

```
paragraph → TEXTITEM * ...  ;
```

which says that a `paragraph` node has zero or more descendants, each belonging to the phylum `TEXTITEM`. That phylum is defined by the lines:

```
TEXTITEM ::= word specialChar formula ...;
word → value is string ;
specialChar → value is string ;
formula → value is MathNotation.ANY ;
```

where `MathNotation` is a separate formalism for mathematical formulae. These lines define our "abstract", "logical", model of text. They say what kinds of things we believe "text" to be comprised of. The `word`, `specialChar`, and `formula` operators are atomic, with their value types specified by the object of the phrase `value is`.

We attach text to structural components of a document with lines such as the following:

```
subsubsection → TITLE PARAGRAPHLIST ;
paragraphList → PARAGRAPH + ...  ;
PARAGRAPHLIST ::= paragraphList ;
PARAGRAPH ::= paragraph ;
title ::= PARAGRAPH ;
```

These say that subsubsection is a binary operator, whose first child is a `TITLE`, and whose second child is a `PARAGRAPHLIST`, i.e., a list of paragraphs. A `TITLE` node must have a `title` operator, which is unary: its child is the `PARAGRAPH` that comprises the text of that title.

Let us briefly consider the `MathNotation` formalism itself (c.f., D. Arnon, S. Mamrak 1991, and D. Arnon, et al., 1988). There, e.g., we define the quotient notation with the lines:

```
quotient → FORMULA FORMULA;
FORMULA ::= quotient sum power ...  name ;
```

Thus quotient is a binary operator.

Thus we use the same definitional mechanism for mathematical formulae as for Articles. As a complete example of the Article formalism, here is an excerpt of a representation of the present paper in it. A more detailed version of this excerpt can be found in the Appendix. Note that we see here how the value of the `formula` operator, which is atomic in the Article formalism, is a term of the `MathNotation` formalism.

```
article[
 header[
  title[
   paragraph[
           word[
"Model-Based Conversions of LaTeX
 Documents "     ]
             ]
        ],
  authors[
     displayItemList[
     paragraph[
       word[ "Dennis S. Arnon
               ... Palo Alto,
                    CA 94304 USA"
                   ]
             ],
                    ]
         ],
  abstract[
    paragraphList[
    paragraph[
        word["Abstract: We are creating"]
         ...
        word[``formulae'' (e.g., " ],
        formula[ quotient[
          diff[ sum[ power[ name[x ],
          number[ 3 ] ], power[ name[ x ],
          number[ 2 ] ] ], number[ 1 ] ],
       ... ] ] ]
]  ]  ]
```

Validation. Validation is the task of deciding whether a given term belongs to a given formalism. Obviously this is a fundamental consistency check that we make frequently when using our system. For example, we take it as a basic criterion for parsing and unparsing steps that the output document, however perturbed it may seem from the input, should

always be valid for the target formalism to which it is supposed to belong.

We may define validation of a term for a formalism by means of a slightly more general notion: the validation of a term t for a phylum P. This we define recursively as follows: let `rootOp` be the operator of the root node of t. If `rootOp` is atomic, then t is valid for P if `rootOp` belongs to P, and if the value of `rootOp` is valid (this involves checking the syntactic correctness of an integer, real, identifier, or string, or recursively checking the validity of its term value for the expected formalism). If `rootOp` is n-Ary, then t is valid for P if `rootOp` belongs to P, if the actual number m of children of `rootOp` is equal to n, and if for $i = 1, ..., n$, $childTerm_i$ is valid for $childPhylum_i$. If `rootOp` is a list operator the definition is similar. Finally, we say that t is valid for the formalism if t is valid for `ROOTPHYLUM`, where `ROOTPHYLUM` consists of the allowable operators of root nodes of terms in this formalism (`ROOTPHYLUM` is analogous to the "start symbol(s)" of a grammar). Often a formalism permits any operator to be a root operator, i.e., its `ROOTPHYLUM` is `ANY`.

Converting Tioga Articles to LaTeX

In this section we first describe the main features of the Tioga editor's document model, and then briefly examine the actual formalism we currently use to model Tioga documents. We then describe the tree pattern matching functions we use to convert Tioga documents to Articles, and finally we outline the actual Tioga-To-Article converter we have written using the tree pattern matching functions. For general background on Tioga, and the Cedar programming environment of which it is a part, the reader may consult Swinehart et al., (1986).

The Tioga document model. Tioga is a true structured document editor in the sense that its internal data structure for any document is a tree of nodes, each of which has character string content. Both entire nodes, and individual characters within a node, can have properties, i.e., attributes. Each node is labelled by an identifier that Tioga calls a *format*. Typical formats are `title`, `abstract`, `head`, `block`, `reference`, etc. Tioga documents are formatted by associating a collection of *style rules* with them. In particular, there should be a style rule for each format that specifies, in a PostScript-like language, how to graphically render nodes of that format. Multiple fonts are provided via both special character properties called *looks*; e.g., `bold`, `italic`, `greek`, plus a mechanism for using the full multinational range of the Xerox character code standard. Mathematics and imbedded illustrations are accommodated via character and node properties.

There is no standard surface syntax for Tioga documents. Virtually all authors of Tioga documents use a small number of standard styles.

A formalism for Tioga documents. As an intermediate step in Tioga to Article conversion, we have defined a formalism which directly expresses the Tioga document model. We call this the *MediumTioga* formalism; we saw an example of it in the subsection on Terms above. We have written Tioga-to-MediumTioga, and MediumTioga-to-Tioga converters. Thus we reduce the Tioga-Article conversion problem to the MediumTioga-Article conversion problem, which we can approach from completely within the Scrimshaw world. Thus, for example, we implement Tioga-to-Article conversion by appropriate tree transformations of Scrimshaw MediumTioga terms to Scrimshaw Article terms.

Tree pattern matching functions. At present our term rewriting capabilities are built on simple tree pattern matching, in which patterns are just literal terms, possibly containing (any number of instances of) pattern variables of two kinds. First, a variable which matches any term (which we call ANYTERM). Second, a variable which matches any list of one or more terms (which we call ANYTERMLIST). For example, here is a list of the patterns we use to search for `title` nodes in MediumTioga documents:

```
leafNode[format[title], ANYTERM]
internalNode[format[title], ANYTERMLIST]
```

Having matched patterns such as the ones above, e.g. having found all the `title` nodes in a MediumTioga document, we then perform a "rewriting" action, e.g. to construct the (unique) title node we must have in the output Article. For this example, the action is to concatenate the text content of all `title` nodes, and their descendant nodes, to make the text content of the Article's title.

Tioga-to-article converter. Our goals are to not fail on any legal Tioga document as input, and to always produce a valid Article as output. Hence, using the pattern matching functions, we traverse the MediumTioga form of an input document and apply a succession of rules to build a valid Article.

Unparsing Articles to LaTeX. An Article term can be unparsed to a LaTeX source file through a straightforward recursive descent tree traversal. A sample of a LaTeX unparsing of an Article representation of this paper is shown in the appendix.

Converting LaTeX Articles to Tioga

The Article document model has been designed to be highly compatible with typical LaTeX representations of documents that are in fact technical articles. Thus at the moment it works well for us to simply define the LaTeX agent model as being identical to the Article user model. Hence at present we "convert" a LaTeX document to an Article by simply "parsing" its LaTeX source file and "recognizing" the Article that we consider to be encoded there. As before, our goals are to not fail on any valid LaTeX input, and to always produce a valid Article as output. Once we have an Article, it is straightforward to unparse it to a MediumTioga document. At present, we ignore unknown control sequences, including macros. We are in the process of developing a separate LaTeX agent model, and upgrading our LaTeX parser to use it.

Bibliography

Arnon, D., R. Beach, K. McIsaac, and C. Waldspurger. "Caminoreal: An Interactive Mathematical Notebook." Pages 1 – 18 in *Proceedings of the International Conference on Electronic Publishing, Document Manipulation, and Typography*, J.C. van Vliet, ed. Cambridge: Cambridge University Press, 1988.

Arnon, D., and S. Mamrak. "On the Logical Structure of Mathematical Notation." *TUGboat* 12(2), pages 479 – 484, 1991.

Borras, P., D. Clément, T. Despeyroux, J. Incerpi, G. Kahn, B. Lang, and V. Pascual. "Centaur: The system." *Proceedings of the* SIGSOFT'88, *Third Annual Symposium on Software Development Environments.* Association for Computing Machinery, Boston, Massachusetts, 1988.

Brown, A., and H. Blair. "A Logic Grammar Foundation for Document Representation and Document Layout." Pages 47 – 64 in *Proceedings of the International Conference on Electronic Publishing, Document Manipulation, and Typography*, R. Furuta, ed. Cambridge: Cambridge University Press, 1990.

Franchi-Zannettacci, P., and D. Arnon "Context-Sensitive Semantics as a Basis for Processing Structured Documents." Pages 135 – 146 in *Proceedings of WOODMAN'89.* Workshop on Object-Oriented Document Manipulation. J. André and Jean Bézivin, editors, (BIGRE 63-64), IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France, Mai 1989, ISSN 0221-5225.

Goldfarb, C. *The SGML Handbook.* Oxford: Clarendon Press, 1990.

Rosenberg, J., M. Sherman, A. Marks, and J. Akkerhuis. *Multi-Media Document Translation: ODA and the EXPRES Project.* New York: Springer-Verlag, 1991.

Swinehart, D., P. Zellweger, R. Beach, and R. Hagmann. "A Structural View of the Cedar Programming Environment." *ACM Transactions on Programming Languages and Systems* 8(4) 419 – 490, 1986.

Appendix

Examples of Conversions

Here is the beginning of the actual Article representation of the Tioga form of this paper.

```
article[
        header[
                title[
                        paragraph[
                                word[
                                        "Model-Based Conversions of LaTeX
Documents "
                                        ]
                                ]
                        ],
                authors[
                        displayItemList[
                                paragraph[
                                        word[
                                                "Dennis S. Arnon"
                                                ],
                                        specialChar[
                                                "(000|012)"
                                                ],
                                        word[
                                                "Xerox PARC"
                                                ],
                                        ....
                                        ]
                                ]
                        ],
                abstract[
                        paragraphList[
                                paragraph[
                                        word[
                                                "Abstract: We are creating a
document conversion .., for example,
                                                ``scientific articles\'\',
``mathematical formulae\'\' (e.g., "
                                                ],
                                        formula[
                                                quotient[
                                                        diff[
                                                                sum[
                                                                        power[

name[

x

],

number[
```

```
3
]
],
                                                                power[
name[
x
],
number[
2
]
]
                                                                        ],
                                                                number[
                                                                        1
                                                                        ]
                                                                ,
                                                                ],
                                                        sum[
                                                                power[
                                                                        name[
x
],
                                                                        number[
2
]
                                                                        ],
                                                                number[
                                                                        1
                                                                        ]
                                                                ]
                                                        ]
                                                ],
                                        word[
                                                "), or ``limericks\'\'.  Each
machine-oriented, ... sidestep
                                                what we believe to be the red
herring of ``markup language\'\'."
                                                ]
                                        ]
                                ]
```

```
                        ],
                keywords[
                        paragraph[
                                word[
                                        "Keywords: Structured Documents,
Electronic Documents, Document Conversion "
                                        ]
                                ]
                        ]
                ],
        body[
                sectionList[
                        sectionIntroOnly[
                                title[
                                        paragraph[
                                                word[
                                                        "Introduction"
                                                        ]
                                                ]
                                        ],
                                paragraphList[
                                        paragraph[
                                                word[
                                                        "In contemporary
document management systems, ...
```

Example of Unparsing

Here is the beginning of the LaTeX unparsing of the document that our system actually produces:

```
\documentstyle[12pt]{Article}
\title{Model-Based Conversions of LaTeX Documents }
\author{
Dennis S. Arnon
%Article% specialChar["(000|012)"]
\\
Xerox PARC
%Article% specialChar["(000|012)"]
\\
3333 Coyote Hill Road
%Article% specialChar["(000|012)"]
\\
Palo Alto, CA 94304 USA
\and
Isabelle Attali
%Article% specialChar["(000|012)"]
\\
INRIA Sophia Antipolis
%Article% specialChar["(000|012)"]
\\
Route des Lucioles
%Article% specialChar["(000|012)"]
\\
06565 Valbonne Cedex, France
```

```
\and
Paul Franchi-Zannettacci
%Article% specialChar["(000|012)"]
\\
CERISI
%Article% specialChar["(000|012)"]
\\
Sophia Antipolis
%Article% specialChar["(000|012)"]
\\
06561 Valbonne Cedex, France }
\begin{document}
\maketitle
\begin{abstract}

Abstract: We are creating a document conversion system based on
...
 for example, ``scientific articles'', ``mathematical formulae'' (e.g.,
${\frac{{{{{{{x}}^{{3}}}}
}+{{{{x}}^{{2}}}}
}}
}-{{1}}}
} {{{{{{x}}^{{2}}}}
}+{{1}}}
}}
$
), or ``limericks''.  Each machine-oriented, or {\em agent} type, is the set
...
 sidestep what we believe to be the red herring of ``markup language''.
\end{abstract}
\begin{keywords}
Keywords: Structured Documents, Electronic Documents, Document Conversion
\end{keywords}
\section{Introduction}



In contemporary document management systems, ...
```

Reports from the 1992 Annual Meeting at Portland

LaTeX 2.09 ↪ LaTeX3: An Update

Chris Rowley

The workshop on LaTeX3 was basically a report on activities of the project in the first half of 1992. This report is an addendum to the article in *TUGboat* (13(1):96 - 101, 1992), which contained a brief sketch of the LaTeX3 Project: its history, its present state and its future, as at the end of 1991. See also *TEX and TUG News* (vol. 1, nos. 1 and 2) for additional updates.

1 The Continuing History

2 Visits and meetings: 1992

March — Workshop in Hamburg

April — Visit to CERN and EP92 conference

April — Alan Hoenig takes on the role of liaison between the project and the TUG Technical Council.

May/June — Meeting of core team in Mainz

June — Special meeting of GUTenberg in Paris.

July — Presentation and mini-workshop at TUG conference in Portland, Oregon.

July — Bowling fund-raiser in Portland.

We are pleased to be able to thank the TUG Board for its unreserved support for the project — and the TUG office for making this support a reality. Without such solid backing, many of our aims would be much more difficult to attain. Since his appointment (and also before it) Alan Hoenig has been enthusiastic in his encouragement and publicity for the project and is now helping us in many ways including the always vital fund-raising which is needed (see below).

The meeting in Mainz was an especially important occasion as it was the first time that the current core team of implementors had all met face-to-face rather than via the megabytes of e-mail correspondence which had been their only previous contact. This potentially traumatic experience was survived by everyone and the outcome was a very useful, intellectually stimulating and enjoyable ten days.

3 Fund-raising

Many thanks must go to Malcolm Clark and Doug Henderson for organising the bowling fund-raiser in Portland: a very enjoyable event at which one of us discovered that the bowls do not seem to go in the right direction as easily as they did 25 years ago. Over $700 was raised by this event — many thanks to all who contributed so generously.

We also wish to thank the European TEX organizations DANTE (Germany) and GUTenberg (France) for their generous contributions and all the many other contributors through whose efforts we have been able to finance important aspects of the work, such as the meeting in Mainz. These include individuals, companies and the national user groups: a list of organisations giving support of various kinds appears regularly in TEX and TUG News.

In addition to the bowling fund-raiser, at the Portland conference we started to look at the possibilities for larger-scale fund-raising from medium-sized and large companies and from trusts and foundations. ETP Services has offered substantial support for this important work. One aspect of this campaign with which *anyone* reading this may be able to help is the supply of information — please contact us if you have any ideas or knowledge about:

- organisations which would be worth canvassing for funds;
- any individuals in medium-to-large companies who would be a good initial contact point for such canvassing.

4 Milestones: 1992

- Clarification of the internal mechanisms needed for parameter handling and their consequences for the processing of environment begin- and end-tags.
- Establishment of the necessity of distinguishing between 'author-defined' environments (and commands) and those environments specified and modified via the style-designer interface.
- Prototype of an enhanced mechanism for passing information from one run of LaTeX to the next.
- Start of a discussion on the design and implementation of the float-handling mechanism. This discussion must involve as wide a range as possible of people involved in typograhic design and typesetting: please contact us if you have experience in this area which could be useful to the project.
- Setting up a network-accessible distribution, maintenance and code-management system for the project at the ZDV, Mainz (this will probably also take over as the source for LaTeX 2.09 system files).
- Release of a new version of LaTeX 2.09 which, in addition to bug fixes, is fully international

(incorporating the functionality of LaTeX) and is fully compatible with the NFSS.

- Setting up a validation system for testing new versions of LaTeX 2.09.

5 Volunteers needed

There are many tasks needing to be done in support of the LaTeX3 project which can be worked on concurrently with the development of the LaTeX3 kernel. Furthermore, some tasks require special expertise not found among the core programming team. Initial research, analysis, and work on these tasks by volunteers can greatly speed up the process of integrating a number of desirable features into LaTeX3. Many of these features can be extensively developed and tested under LaTeX 2.09 even before the LaTeX3 kernel is available.

Therefore a list of volunteer tasks has been drawn up, in the form of a LaTeX article, which will shortly (probably by the time you read this) be circulated as widely as possible to the LaTeX user community through various channels: mail lists such as `TeXhax`, `Info-TeX`, `Euro-TeX`; newsgroups such as `comp.text.tex`; anonymous FTP and mail servers from major TeX archives; and publication in print via *TUGboat* and any other journals and newsletters that are interested to print it.

6 Bibliography

Frank Mittelbach and Chris Rowley LaTeX2.09 ↪ LaTeX3. *TUGboat*, 13(1):96–101, 1992.

Workshop Summary

TeX Archives

Peter Abbott

Although not all the major archives were represented at the meetings, it was felt that the following would be of advantage to the world community.

- A site (not necessarily an archive) will 'own' an item of software. That site will be the definitive source for the current version of that item. Archives will collect/receive items from the 'owner'.
- Every attempt will be made to keep the archives in step and up to date.
- Authors will be requested to use standard header formats in ASCII files, details of which will be circulated later.
- `Read.Me` files for collections of PK font files will contain header details stating from which MF sources they were generated (e.g., Aston will move towards holding 300 dpi PK files for Canon SX (write black), 180 dpi, and 240 dpi).
- Aston may (probably will) move to a UNIX machine but VMS binaries will be retained and, if possible, a VMS-like interface will be provided.
- Due notice will be taken of directory and filenames to prevent them exceeding 80 characters in total if possible, since many mailers will truncate long lines.
- Case of letters in names should be irrelevant.

Aston has also undertaken to make available

- WAIS — Wide Area Information Server;
- Gopher — The Internet Gopher Service;
- ARCHIE — Archie entries (A VMS client has been announced); and
- WWW (W^3) — World Wide Web.

Aston currently has its directory available for search by WAIS (i.e., the ability to locate any file name immediately). UKTeX and TeXhax are also available in indexed form.

Aston is aiming to make available a front end to link the synthetic catalogues such as David Jones' to the actual files in the archive. The same systems can be used to access more developed books, such as *The TeXbook*.

Aston already has a crude model of activity for WAIS and TeX files, whereby the `*.dvi*` are indexed word by word. The user is returned a piece of `dvi` representing a printed page.

Workshop Summary

Getting PostScript into TeX and LaTeX Documents

Anita Z. Hoover

Approximately 65 people came to learn and contribute ideas on how to include PostScript files into TeX and LaTeX using `dvips`, the popular DVI→PostScript driver written by Tomas Rokicki of Stanford University.

The basic objectives covered:

1. What is a Bounding Box?
   Tells how big the graphic is and where it is located on the page. It represents the lower-left and upper-right corners of a box which would surround the graphic in the PostScript file.
2. What if I don't have a Bounding Box?
   You need to use `bbfig` or some other program or calculate it by hand.
3. What is the page orientation for PostScript?
   Looking at a portrait page, the lower-left corner is the origin (0,0) and the upper-right corner is

(612,792) for American letter size paper. The upper-right corner will be different for other paper sizes.

4. How do I include my PostScript pictures?
   Two macro packages are available
   - **epsf**
     TEX:
     `\input epsf`
     LATEX:
     `\documentstyle [epsf] {style}`
   - **psfig**
     TEX:
     `\input psfig`
     LATEX:
     `\documentstyle {style}`
     `\input{psfig}`

A document was distributed that explained in detail how to include PostScript in TEX and LATEX documents. This was specifically written for the UNIX environment at the University of Delaware, but can easily be applied to all computer environments.

Eight examples were distributed that showed specific PostScript files being included from a variety of applications such as Macintosh CricketGraph, Mathematica, Framemaker, WordPerfect, Lotus, SAS Graphics, Macintosh MacDrawII, and Macintosh SuperPaint.

All examples and documentation are available via anonymous ftp from

`zebra.cns.udel.edu (128.175.8.11) in pub/tex/workshoptug92/PostScript.`

Workshop Summary

LATEX: How to Use Style Options

Anita Z. Hoover

Approximately 45 people came to learn and contribute ideas on how to use specific style options available with LATEX. Many of the names of style files needed to be reduced to 8-character file names with 3-character extensions. As a result the original name taken off the archive is listed in parenthesis.

1. How to rotate a table (based on using `dvips` written by Tomas Rokicki).
   - rotate.sty/rotate.tex
   - nrotate.sty (newrotate.sty)
2. How to create two-up pages without using PostScript.
   - 2up.sty
3. How to create a draft overlay for every page using PostScript (based on using `dvips` written by Tomas Rokicki).
   - drpshead.sty `\draftcp`
   - draftps.sty `\draft`
4. How to produce tables/figures side by side.
   - Use the minipage environment.
5. How to modify your headings and footers.
   - fancyhd.sty (fancyheadings.sty)
6. How to number your equations, 1, 2a, 2b, 2c, 3, 4a, 4b, 5, etc.
   - subeqn.sty
   - subeqnar.sty (subeqnarray.sty)
7. How to continue a table across multiple pages without having to do so by hand.
   - bigtab.sty (bigtabular.sty)
   - supertab.sty (supertabular.sty)

All examples and documentation are available via anonymous ftp from

`zebra.cns.udel.edu (128.175.8.11) in pub/tex/workshoptug92/LaTeXstyles.`

Workshop Summary

Modifying manmac to Suit the Publisher

Dan Olson

This was the second year for this workshop to be presented in conjunction with the TUG Annual Meeting. It is not intended as an introduction to macro writing. Instead, it is an introductory look at the inner workings of macro packages. We looked at modifications made to the `manmac` macros (used in producing *The TEXbook*) to implement common publisher requests. Topics explained included modifying the fonts used in a document, changing the page size, and reformatting the running heads, footnotes, and exercises. Examples included the use of such TEX commands as `\llap`, `\rlap`, and `\hangindent`. The attendees had Macintoshes running Textures available for hands-on work.

Participants at the 13th Annual TUG Meeting
July 27–30, 1992 Portland, Oregon

* indicates an exhibitor

Peter Abbott
Aston University
Birmingham, England, U.K.

Robert A. Adams
University of British Columbia
Vancouver
British Columbia, Canada

Clifford Alper
TEX Users Group
Providence, Rhode Island

Dennis Arnon
Xerox Palo Alto Research Center
Palo Alto, California

Harry Baldwin
San Diego City College
San Diego, California

Frederick H. Bartlett
Bartlett Press Incorporated
Somerset, New Jersey

Charles W. Beardsley
American Society of Mechanical Engineers
New York, New York

Nelson H. F. Beebe
University of Utah
Salt Lake City, Utah

Barbara Beeton
American Mathematical Society
Providence, Rhode Island

Larry Bennett
South Dakota State University
Brookings, South Dakota

Chris Biemesderfer
National Optical Astronomy Observatories
Tucson, Arizona

Angelika Binding
Springer-Verlag Heidelberg
Heidelberg, Germany

Antonio Bockh
Intevep
Caracas, Venezuela

Kathy Borg-Todd
University of South Carolina
Columbia, South Carolina

Johannes Braams
PTT Research Neher Laboratorium
Leidschendam
The Netherlands

Mimi Burbank
Florida State University
Tallahassee, Florida

William P. Butler
TEX Users Group
Providence, Rhode Island

Katherine Butterfield
University of California, Berkeley
Berkeley, California

Stephen Carlson
Williamsburg, Virginia

Christopher Carruthers
University of Ottawa
Ottawa, Ontario, Canada

Katharine S. Carter
Princeton University
Princeton, New Jersey

William Casselman
University of British Columbia
Vancouver
British Columbia, Canada

S. Bart Childs
Texas A&M University
College Station, Texas

Daniel Christiansen
Albion College
Albion, Michigan

Malcolm W. Clark
Polytechnic of Central London
London, England, U.K.

A.C. Conrad
Menil Foundation
Houston, Texas

* **Betsy J. Dale**
ArborText Incorporated
Ann Arbor, Michigan

Heather Dalterio
American Astronomical Society
Washington, DC

Jackie Damrau
Superconducting Super Collider Laboratory
Dallas, Texas

Donald W. DeLand
Integre Technical Publishing Company
Albuquerque, New Mexico

Christine Detig
Technische Universität Darmstadt
Darmstadt, Germany

Luzia Dietsche
Universität Heidelberg
Heidelberg, Germany

Andrea Domst
State University of New York at Fredonia
Fredonia, New York

Faith Donaldson
TSI Graphics
Effingham, Illinois

Michael Doob
University of Manitoba
Winnipeg, Manitoba
Canada

Michael J. Downes
American Mathematical Society
Providence, Rhode Island

* **Ken Dreyhaupt**
Springer-Verlag New York Inc.
New York, New York

Rochelle Edmond
NASA Ames Research Center
Moffett Field, California

Victor L. Eijkhout
University of Tennessee, Knoxville
Knoxville, Tennessee

Michael J. Ferguson
Université du Québec
Verdun, Québec, Canada

Peter Flynn
University College of Cork
Cork, Republic of Ireland

Jim Fox
University of Washington
Seattle, Washington

Harumi Fujiura
ASCII Corporation
Kawasaki, Japan

Edward A. Garay
University of Illinois at Chicago
Chicago, Illinois

Mary Louise Garcia
Los Alamos National Laboratory
Los Alamos, New Mexico

Ron Gardner
Gardner Indexing Service
Edmonton, Alberta, Canada

* **Doug Garnett**
Blue Sky Research
Portland, Oregon

Helen M. Gibson
Wellcome Institute for the History of Medicine
London, England, U.K.

* **Regina Girouard**
American Mathematical Society
Providence, Rhode Island

Philip Goldstein
Space Telescope Science Institute
Baltimore, Maryland

Raymond E. Goucher
Detroit, Michigan

Geeti Granger
John Wiley & Sons Ltd.
Chichester, England, U.K.

Paula Gudder
Denver, Colorado

Rakesh Gupta
Techbooks Incorporated
Fairfax, Virginia

William L. Haberman
Rockville, Maryland

James Hafner
San Jose, California

Christopher Hamlin
American Institute of Physics
Woodbury, New York

Yannis Haralambous
Villeneuve d'Ascq, France

* **Robert L. Harris**
Micro Programs Incorporated
Syosset, New York

Richard N. Hayes
ETP Services Co.
Portland, Oregon

Doug Henderson
Blue Sky Research
Portland, Oregon

* **Amy Hendrickson**
TEXnology Incorporated
Brookline, Massachusetts

Karin Hendrickson
University of Washington
Seattle, Washington

Matthew N. Hendryx
Academic Graphics Typography
North Manchester, Indiana

Robert H. Hilbert
John Wiley & Sons Incorporated
New York, New York

John D. Hobby
AT&T Bell Laboratories
Murray Hill, New Jersey

Alan Hoenig
John Jay College, City University of New York
New York, New York

Stephanie Hogue
University of Pennsylvania
Philadelphia, Pennsylvania

Anita Z. Hoover
University of Delaware
Newark, Delaware

* **Berthold Horn**
Y & Y
Carlisle, Massachusetts

* **Blenda Horn**
Y & Y
Carlisle, Massachusetts

* **Don Hosek**
Quixote Digital Typography
Claremont, California

Calvin W. Jackson
California Institute of Technology
Pasadena, California

Mimi Jett
ETP Services Co.
Portland, Oregon

Gordon C. Johnson
Interactive Composition Corporation
Pleasant Hill, California

Thomas Judson
University of Portland
Portland, Oregon

Becky Kaluza
Blue Sky Research
Portland, Oregon

David Kellerman
Northlake Software
Portland, Oregon

* **Linda King**
ArborText Incorporated
Ann Arbor, Michigan

Timo Knuutila
University of Turku
Turku, Finland

Kresten Krab Thorup
Aalborg, Denmark

David H. Kratzer
Los Alamos National Laboratory
Los Alamos, New Mexico

Candy Lafrenz
ETP Services Co.
Portland, Oregon

Francois Lambert
Université de Montréal
Montréal, Québec, Canada

Joachim Lammarsch
Universität Heidelberg
Heidelberg, Germany

Timothy R. Larson
Behrend College
Erie, Pennsylvania

Dan C. Latterner
Mathematical Reviews
Ann Arbor, Michigan

Warren Leach
Blue Sky Research
Portland, Oregon

Dan Levin
Educaide Software
Vallego, California

Silvio Levy
University of Minnesota
Minneapolis, Minnesota

Frank Lusardi
New York, New York

Pierre MacKay
University of Washington
Seattle, Washington

David Marks
Imperial College
London, England

Elizabeth McCarthy
IBM T. J. Watson Research Center
Yorktown Heights
New York

Robert W. McGaffey
Martin Marietta Energy Systems Inc.
Oak Ridge, Tennessee

Wendy McKay
Université de Montréal
Montréal, Québec, Canada

Carol A. Meyer
Association for Computing Machinery, Inc.
New York, New York

Lothar Meyer-Lerbs
Bremen, Germany

Cornelia M. Monahan
American Society of Mechanical Engineers
New York, New York

Patricia Monohon
University of California, Santa Barbara
Santa Barbara, California

Mary Jean Moore
University of California, Oakland
Oakland, California

Norman Naugle
Texas A&M University
College Station, Texas

Florence Neuberger
East Setauket, New York

Arthur Ogawa
Palo Alto, California

* **Daniel D. Olson**
ETP Services Co.
Portland, Oregon

* **Fred Osborne**
TCI Software Research, Inc.
Las Cruces, New Mexico

Yoko Ozawa
NEC Research Institute, Inc.
Princeton, New Jersey

* **George Pearson**
TCI Software Research, Inc.
Las Cruces, New Mexico

Brett Perkes
Interactive Composition Corporation
Logan, Utah

Bill Pikounis
Merck Sharp and Dohme Research Laboratories
Rahway, New Jersey

Craig R. Platt
University of Manitoba
Winnipeg, Manitoba
Canada

Jon Radel
Reston, Virginia

Pat Radnich
Personal TEX Incorporated
Mill Valley, California

T. V. Raman
Cornell University
Ithaca, New York

Maria Ramirez
University of Washington
Seattle, Washington

Cynthia Rodriguez
University of Illinois at Chicago
Chicago, Illinois

Chris Rowley
The Open University
London, England, U.K.

Beverly J. Ruedi
Mathematical Association of America
Washington, DC

Jan Michael Rynning
Stockholm, Sweden

David Salomon
California State University, Northridge
Northridge, California

Joachim Schrod
Technische Universität Darmstadt
Darmstadt, Germany

John T. Sheridan
Sheridan Printing Systems, Inc.
Alpha, New Jersey

* **Barry Smith**
Blue Sky Research
Portland, Oregon

Jenny Smith
John Wiley & Sons Ltd.
Chichester, England, U.K.

Lee Smith
ETP Services Co.
Portland, Oregon

Lowell Smith
Salt Lake City, Utah

Michael Sofka
Publication Services
Champaign, Illinois

Friedhelm Sowa
Heinrich Heine University
Düsseldorf, Germany

Anthony Starks
Newark, New Jersey

David K. Steiner
Rutgers University
Piscataway, New Jersey

Christina Thiele
Carleton Production Centre
Nepean, Ontario, Canada

Marlene Thom
Brooks/Cole Publishing Company
Pacific Grove, California

Margaret Thomas
Talaris Systems Inc.
San Diego, California

Lee F. Thompson
University of Wisconsin, Madison
Madison, Wisconsin

Thomas M. Thompson
Walla Walla College
College Place, Washington

Andy Trinh
Beckman Instruments Incorporated
Brea, California

Frank H. Ulmer
Grumman Melbourne Systems Division
Melbourne, Florida

Walter van der Laan
PTT Research Neher Laboratorium
Leidschendam
The Netherlands

Margaret L. Ward
Massachusetts Institute of Technology
Cambridge, Massachusetts

Stacy Waters
University of Washington
Seattle, Washington

Neil A. Weiss
Arizona State University
Tempe, Arizona

Alan Wetmore
U.S. Army Atmospheric Sciences Laboratory
White Sands Missile Range
New Mexico

Ron Whitney
TEX Users Group
Providence, Rhode Island

William B. Woolf
American Mathematical Society
Providence, Rhode Island

* **Ralph E. Youngen**
American Mathematical Society
Providence, Rhode Island

The Donald E. Knuth Scholarship: 1992 Scholar and 1993 announcement

Jenny Smith, an employee of the Production Department of John Wiley and Sons, Ltd., Chichester, England, was honored at the 1992 TUG Annual Meeting in Portland, Oregon, as the 1992 Donald E. Knuth Scholar. Ms. Smith is the seventh recipient of the Scholarship. The award included all expenses associated with attendance at the meeting, and at the short course on advanced TEX and macro writing which followed it.

The intent of the Knuth Scholarship is to encourage the increase of knowledge about TEX and to sharpen the TEX skills of non-technical users.

The Committee reported that all entries showed good to excellent standards of typography. The major distinction of the winner's project was to demonstrate skills in production of a wide range of documents, including modifying and writing macros for some of the documents.

Thanks are extended to the 1992 Scholarship Committee. Chris Rowley (chair), and David Salomon will remain on the Committee for 1993. Leaving the Committee are Nico Poppelier, who will become the liaison between the Committee and the TUG Board of Directors, and Linda Williams, the 1991 Knuth Scholar. Jenny Smith will join the Committee for one year in her role as the current Knuth Scholar.

Announcement of the 1993 competition

One Knuth Scholarship will be available for award next year. The competition will be open to all 1993 TUG members holding support positions with duties that are secretarial, clerical or editorial in nature. It is therefore not intended for those with a substantial training in technical, scientific or mathematical subjects and, in particular, it is not open to anyone holding, or studying for, a degree with a major or concentration in these areas.

The award will consist of an expense-paid trip to the 1993 TUG Annual Meeting at Aston University in Birmingham, England, and to the Scholar's choice from the short courses offered in conjunction with that meeting. A cap of $2000 has been set for the award; however, this does not include the meeting or course registration fees, which will be waived.

To enter the competition, applicants should submit to the Scholarship Committee, by the deadline specified below, the input file and final TEX output of a project that displays originality, knowledge of TEX, and good TEXnique.

The project as submitted should be compact in size. If it involves a large document or a large number of documents then only a representative part should be submitted, together with a description of the whole project. For example, from a book just one or two chapters would be appropriate.

The project may make use of a macro package, either a public one such as LATEX or one that has been developed locally; such a macro package should be identified clearly. Such features as sophisticated use of math mode, of macros that require more than "filling in the blanks", or creation and use of new macros will be taken as illustrations of the applicant's knowledge.

All macros created by the candidate should be well documented with clear descriptions of how they should be used and an indication of how they work internally.

All associated style files, macro-package files, etc., should be supplied, or a clear indication given of any widely available ones used (including version numbers, dates, etc.); clear information should be provided concerning the version of TEX used and about any other software (e.g. particular printer drivers) required. Any nonstandard fonts should be identified and provided in the form of `.tfm` and `.pk` files suitable for use on a 300dpi laser printer.

While the quality of the typographic design will not be an important criterion of the judges, candidates are advised to ensure that their printed output adheres to sound typographic standards; the reasons for any unusual typographic features should be clearly explained.

All files and documents comprising the project must be submitted on paper; the input files should be provided in electronic form as well. Suitable electronic media are IBM PC-compatible or Macintosh diskettes, or a file sent by electronic mail.

Copies of this announcement and some additional information are available from the TUG office. To obtain a copy, or to request instructions on e-mail submission, write to the address at the end of this announcement, or send a message by e-mail to `TUG@Math.AMS.org` with the subject "Knuth Scholarship request".

Along with the project, each applicant should submit a letter stating the following:

1. affirmation that he/she will be available to attend the 1993 annual meeting;
2. affirmation of willingness to participate on the committee to select the next Scholar.

Each applicant should also submit a *curriculum vitae* summarizing relevant personal information, including:

1. statement of job title, with a brief description of duties and responsibilities;
2. description of general post-secondary school education, TEX education, identifying courses attended, manuals studied, personal instruction from experienced TEX users, etc.;
3. description of TEX resources and support used by the candidate in the preparation of the project.

Neither the project nor the *curriculum vitae* should contain the applicant's name or identify the applicant. These materials will be reviewed by the committee without knowledge of applicants' identities. If, despite these precautions, a candidate is identifiable to any judge, then that judge will be required to make this fact known to the others and to the TUG board members responsible for the conduct of the judging.

The covering letter, *curriculum vitae*, and all macro documentation that is part of the project input should be in English. (English is not required for the output of the project.) However, if English is not the applicant's native language, that will not influence the decision of the committee.

Selection of the Scholarship recipient will be based on the project submitted.

Schedule

The following schedule will apply; all dates are in 1993:

March 1	Deadline for receipt of submissions
March 15–May 10	Judging period
May 17	Notification of winner
July 26–29	1993 Annual Meeting, Birmingham, England

The 1993 Scholarship Committee consists of

- Chris Rowley, Open University, UK (Chair);
- David Salomon, California State University, Northridge;
- Jenny Smith, John Wiley and Sons, Ltd., Chichester, England.

Where to write

All applications should be submitted to the Committee in care of the TUG office:

TEX Users Group
Attn: Knuth Scholarship Competition
653 North Main Street
P. O. Box 9506
Providence, RI 02940-9506
U.S.A.
email: `TUG@math.ams.org`

◇ Barbara Beeton
Liaison to the 1992 Committee

Calendar

1992

Nov 17	**TUGboat Volume 14, 1st regular issue:** Deadline for receipt of *technical* manuscripts.
Nov 19	NTG Fall Meeting, "LATEX, LATEX 3, and font selection", Meppel (near Groningen), The Netherlands. For information, contact Gerard van Nes (`vannes@ECN.NL`).
Nov 24	**TUGboat Volume 13, 3rd regular issue:** Mailing date (tentative).
Dec 15	**TUGboat Volume 14, 1st regular issue:** Deadline for receipt of news items, reports.

1993

Feb	UK TEX Users' Group, London. Topic: Front ends for TEX; how successful are the WYSIWYG packages for non-TEX users and for wizards? For information, contact Carol Hewlett (`hewlett@vax.lse.ac.uk`).
Feb 16	**TUGboat Volume 14, 2nd regular issue:** Deadline for receipt of *technical* manuscripts (tentative).

Status as of 15 September 1992

Feb	23	TEX for Publishers, Boston, Massachusetts.
Feb	24–27	CONCEPPTS 93, The Prepublishing Conference, Orange County Convention Center, Orlando, Florida. "International Conference on Computers and Electronic Publishing and Printing Technologies". For information, phone: 703-264-7200, Fax: 703-620-9187.

San Francisco, California

Mar	1–5	Intensive LATEX
Mar	8–9	Practical SGML and TEX

Mar	9	**TUGboat Volume 14, 1st regular issue:** Mailing date (tentative).
Mar	9–12	DANTE'93 and General Meeting, Chemnitz, Germany. For information, contact Dr. Wolfgang Riedel (`wolfgang.riedel@hrz.tu-chemnitz.de`).
Mar	16	**TUGboat Volume 14, 2nd regular issue:** Deadline for receipt of news items, reports (tentative).
Mar		UK TEX Users' Group, Glasgow, Scotland. (Two days, just before the BCS EPSG meeting; postponed from April 1992.) Topics: METAFONT, theoretical and practical; and font selection schemes, virtual fonts, multiple languages and hyphenation, etc.—everything you need to know to use TEX to typeset foreign languages. For information, contact Carol Hewlett (`hewlett@vax.lse.ac.uk`).
Apr	14	TEX for Publishers, Washington, DC.
Apr	19–23	Beginning/Intermediate TEX, Boston, Massachusetts.
Apr	26–30	Intensive LATEX, Boston, Massachusetts.
May		UK TEX Users' Group, Chichester, England. Visit to John Wiley & Sons Ltd. Host: Geeti Granger. For information, contact Carol Hewlett (`hewlett@vax.lse.ac.uk`).
May	25	**TUGboat Volume 14, 2nd regular issue:** Mailing date (tentative).
May-Jun		NTG Spring Meeting, "Typography in past and future" [location to be announced]. For information, contact Gerard van Nes (`vannes@ECN.NL`).
Jun	6–9	Society for Technical Communication, 40th Annual Conference. Dallas, Texas. For information, contact the Society headquarters, 901 N. Stuart St., Suite 304, Arlington, VA 22203-1822. (703-522-4114; Fax: 703-522-2075) Proposals for presentations due by August 1, 1992.
Jun	9	TEX for Publishers, New York City.

San Diego, California

Jun	7–11	Modifying LATEX Style Files
Jun	14–19	Beginning/Intermediate TEX
Jun	21–25	Advanced TEX and Macro Writing

Aug	9–13	Beginning/Intermediate TEX, Boston, Massachusetts
Aug	17	**TUGboat Volume 14, 3rd regular issue:** Deadline for receipt of *technical* manuscripts (tentative).
Aug	23–27	Intensive LATEX, Ottawa, Canada.
Sep	14	**TUGboat Volume 14, 3rd regular issue:** Deadline for receipt of news items, reports (tentative).
Sep	22	TEX for Publishers, Orlando, Florida

Boston, Massachusetts

Sep	23–24	Book and Document Design with TEX
Oct	25–29	Intensive LATEX
Nov	1–5	Advanced TEX and Macro Writing
Nov	8–9	Practical SGML and TEX

Oct	18–22	Beginning/Intermediate TEX, Chicago, Illinois
Nov	23	**TUGboat Volume 14, 3rd regular issue:** Mailing date (tentative).

For additional information on the events listed above, contact the TUG office (401-751-7760, email: `tug@math.ams.com`) unless otherwise noted.

TUG93 call for papers

World Wide Window on TEX

14th Annual TEX Users Group Meeting July 26th – 29th, 1993

Aston University in Birmingham, UK, will be the venue for the 1993 TUG conference. Aston is the home of the 'Aston Archive', one of the largest collections of electronic TEX paraphernalia. This is the first time the annual meeting will have been held outside North America.

The location of the conference at one centre of the electronic web and its movement from North America encourages particular focus on the 'world-wide' aspects of TEX (LATEX, METAFONT...). The marked rise in maturity of windowing systems (Macintosh, Atari, Amiga, Windows3, X windows) also allows us to exploit more straightforward and direct ways of employing the TEX tools. It is hoped that there will be a contribution to the conference from the Didot project, further extending the range of topics to include digital typography and font creation.

The conference will feature the regular paper presentations, but workshops, poster displays, courses, panels and 'birds of a feather' sessions will be integral components.

Contributions are being actively sought in the following subject areas: ⋄ archives ⋄ electronic networks ⋄ formatting structured documents ⋄ LATEX3 ⋄ graphical user interfaces to TEXware ⋄ non-english issues ⋄ non-Latin scripts ⋄ digital typography ⋄ editing structured documents ⋄ styles ⋄ other typesetting systems ⋄ document views ⋄

Program coordinators

	Chris Rowley	Malcolm Clark
	Parsifal College	IRS
	Open University	University of Westminster
	Finchley Road	115 New Cavendish Street
	London NW3 7BG	London W1M 8JS
phone:	071 794 0575	071 911 5000 ex 3622
email:	`ca_rowley@uk.ac.open.acs.vax`	`malcolmc@uk.ac.wmin`
fax:	071 433 6196	071 911 5093

Conference committee

Peter Abbott, Chris Rowley, Philip Taylor, Carol Hewlett, Sebastian Rahtz, David Osborne, Malcolm Clark

TEX Users Group
1993 Course Schedule

Beginning/Intermediate TEX

Boston	April 19–23
San Diego	June 14–19
Boston	August 9–13
Chicago	October 18–22

Intensive Course in LATEX

San Francisco	March 1–5
Boston	April 26–30
Ottawa	August 23–27
Boston	October 25–29

TEX for Publishers

Boston	February 23
Washington, DC	April 14
New York City	June 9
Orlando	September 22

Modifying Latex Style Files

San Diego	June 7–11

Advanced TEX and Macro Writing

San Diego	June 21–25
Boston	November 1–5

Practical SGML and TEX

San Francisco	March 8–9
Boston	November 8–9

Book and Document Design with TEX

Boston	September 23–24

With the exception of *TEX for Publishers* and *Book and Document Design with TEX*, each class will be run in laboratory style with computers provided for all students.

TUG courses are small with 8 to 15 students in most classes and are held at major hotels.

The dates and locations above are tentative — watch for the final schedule in the mail this fall.

For more information about these courses, contact the TEX Users Group at (401) 751-7760.

---

and ...

On-site courses in TEX and LATEX are also available from the TEX Users Group

Courses in TEX, LATEX, SGML and TEX, PostScript, or TEX for Publishers at every level, from beginning to advanced, can be arranged and tailored to your needs. The course fee includes all instructor fees and expenses plus textbooks and other materials for up to 15 students. If a properly equipped training facility is not available, TUG will arrange computer rentals and supply TEX or LATEX software.

Contact the TEX Users Group at (401) 751-7760 for more information.

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

Beckman Instruments,
Diagnostic Systems Group,
Brea, California

Belgrade University,
Faculty of Mathematics,
Belgrade, Yugoslavia

Brookhaven National Laboratory,
Upton, New York

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

CERN, *Geneva, Switzerland*

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

Cornell University,
Mathematics Department,
Ithaca, New York

DECUS, Electronic Publishing
Special Interest Group,
Marlboro, Massachusetts

Department of National Defence,
Ottawa, Ontario, Canada

Digital Equipment Corporation,
Nashua, New Hampshire

E. S. Ingenieres 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

GKSS, Forschungszentrum
Geesthacht GmbH,
Geesthacht, Federal Republic of
Germany

Grinnell College,
Computer Services,
Grinnell, Iowa

Grumman Aerospace,
Melbourne Systems Division,
Melbourne, Florida

GTE Laboratories,
Waltham, Massachusetts

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

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

Politecnico di Torino,
Torino, Italy

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.

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 California, Santa Barbara, *Santa Barbara, California*

University of Canterbury,
Christchurch, New Zealand

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 Massachusetts,
Amherst, Massachusetts

Università degli Studi di Trento,
Trento, Italy

University of Oslo,
Institute of Informatics,
Blindern, Oslo, Norway

University of Oslo,
Institute of Mathematics,
Blindern, Oslo, Norway

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

Villanova University,
Villanova, Pennsylvania

Virginia Polytechnic Institute,
Interdisciplinary Center for Applied Mathematics,
Blacksburg, Virginia

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

TEX CONSULTING & PRODUCTION SERVICES

North America

Abrahams, Paul

214 River Road, Deerfield, MA 01342; (413) 774-5500

Development of TEX macros and macro packages. Short courses in TEX. Editing assistance for authors of technical articles, particularly those whose native language is not English My background includes programming, computer science, mathematics, and authorship of *TEX for the Impatient.*

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.

1000 Victors Way, Suite 400, Ann Arbor, MI 48108; (313) 996-3566

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.

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.

NAR Associates

817 Holly Drive E. Rt. 10, Annapolis, MD 21401; (410) 757-5724

Extensive long term experience in TEX book publishing with major publishers, working with authors or publishers to turn electronic copy into attractive books. We offer complete free lance production services, including design, copy editing, art sizing and layout, typesetting and repro production. We specialize in engineering, science, computers, computer graphics, aviation and medicine.

Ogawa, Arthur

1101 San Antonio Road, Suite 413, Mountain View, CA 94043-1002; (415) 691-1126; `ogawa@applelink.apple.com`.

Specialist in fine typography, LATEX book production systems, database publishing, and SGML. Programming services in TEX, LATEX, PostScript, SGML, DTDs, and general applications. Instruction in TEX, LATEX, and SGML. Custom fonts.

Pronk&Associates Inc.

1129 Leslie Street, Don Mills, Ontario, Canada M3C 2K5;

(416) 441-3760; Fax: (416) 441-9991

Complete design and production service. One, two and four-color books. Combine text, art and photography, then output directly to imposed film. Servicing the publishing community for ten years.

Quixote Digital Typography, Don Hosek

349 Springfield, #24, Claremont, CA 91711; (714) 621-1291

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 to meet publisher's or designer's specifications; instruction.

Type 2000

16 Madrona Avenue, Mill Valley, CA 94941;
(415) 388-8873; FAX (415) 388-8865

$2.50 per page for 2000 DPI TEX camera ready output! We have a three year history of providing high quality and fast turnaround to dozens of publishers, journals, authors and consultants who use TEX. Computer Modern, Bitstream and METAFONT fonts available. We accept DVI files only and output on RC paper. $2.25 per page for 100+ pages, $2.00 per page for 500+ pages.

Outside North America

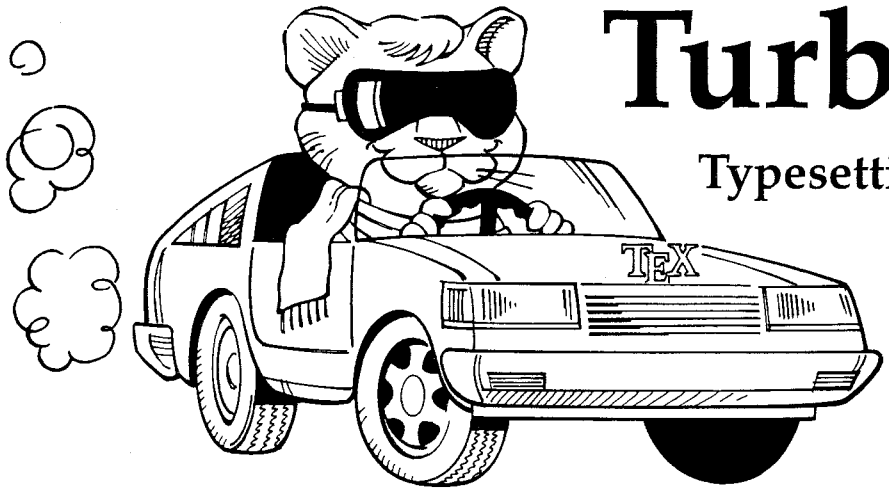
TypoTEX Ltd.

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.

Information about these services can be obtained from:

TEX Users Group
P. O. Box 9506
Providence, RI 02940
(401) 751-7760



TurboTEX

Typesetting Software

Executables $150
With Source $300

MICROSOFT® WINDOWS™ COMPATIBLE

THE MOST VERSATILE TEX ever published is breaking new ground in the powerful and convenient graphical environment of Microsoft Windows: TurboTEX Release 3.1E. TurboTEX runs on all the most popular operating systems (Windows, MS-DOS, OS/2, and UNIX) and provides the latest TEX 3.14 and METAFONT 2.7 standards and certifications: preloaded plain TEX, LATEX, AMS-TEX and AMS-LATEX, previewers for PC's and X-servers, METAFONT, Computer Modern and LATEX fonts, and printer drivers for HP LaserJet and DeskJet, PostScript, and Epson LQ and FX dot-matrix printers.

■ **Best-selling Value:** TurboTEX sets the world 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 around the globe 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. 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 interface; 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 versions run on Microsoft Windows 3.0 or 3.1. Order source code (includes Programmer's Guide) for other machines. On the PC, source compiles with Microsoft C, Watcom C 8.0, or Borland C++ 2.0; other operating systems need a 32-bit C compiler supporting UNIX standard I/O. Specify 5-1/4" or 3-1/2" PC-format floppy disks.

■ **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-TEX Fonts

TEX-compatible Bit-Mapped Fonts Identical to Adobe PostScript Typefaces

If you are hungry for new TEX fonts, here is a feast guaranteed to satisfy the biggest appetite! The AP-TEX fonts serve you a banquet of gourmet delights: 438 fonts covering 18 sizes of 35 styles, at a total price of $200. The AP-TEX fonts consist of PK and TFM files which are exact TEX-compatible equivalents (including "hinted" pixels) to the popular PostScript name-brand fonts shown at the right. Since they are directly compatible with any standard TEX implementation (including kerning and ligatures), you don't have to be a TEX 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

Avant Garde Bold
Avant Garde Bold Oblique
Avant Garde Demibold
Avant Garde Demibold Oblique
Bookman Light
Bookman Light Italic
Bookman Demibold
Bookman Demibold Italic
Courier
Courier Oblique
Courier Bold
Courier Bold Oblique
Helvetica
Helvetica Oblique
Helvetica Bold
Helvetica Bold Oblique
Helvetica Narrow
Helvetica Narrow Oblique
Helvetica Narrow Bold
Helvetica Narrow Bold Oblique
Schoolbook New Century Roman
Schoolbook New Century Italic
Schoolbook New Century Bold
Schoolbook New Century Bold Italic
Palatino Roman
Palatino Italic
Palatino Bold
Palatino Bold Italic
Times Roman
Times Italic
Times Bold
Times Bold Italic
Zapf Chancery Medium Italic
Symbol ΔΦΓϑΛΠΘ
Zapf Dingbats ✂☞❑

Index of Advertisers

407	American Mathematical Society
404	ArborText
Cover 3	Blue Sky Research
405	ETP (Electronic Technical Publishing)
410	K-Talk Communications
408, 409	Kinch Computer Company
403	Life Science Communications Ltd.
407	Micro Programs, Inc.
411	Personal TEX Inc.
406	Springer-Verlag
412	Y&Y

TEX *without* Bitmaps

Wouldn't it be nice to be able to preview DVI files at any magnification, not just those for which bitmap fonts have been pre-built? Or to produce truly resolution-independent output that will run on any PostScript device, whether image setter or laser printer?

Perhaps you are looking for an alternative to Computer Modern? There now exist complete outline font sets which include math fonts that are direct replacements for those in CM. Even if you do want to remain faithful to CM, there are distinct advantages to switching to the outline version of the fonts. We supply the tools to do all of this:

DVIWindo — *preview DVI files calling for outline fonts*

* Preview at arbitrary magnification
* Preview in MS Windows™ — a simple, standardized user interface
* Print to any printer with a Windows printer driver
* Show EPSF files with preview on screen — and insert TIFF images

DVIPSONE — *partial font downloading for speed and efficiency*

* Avoid running out of memory on the printer
* Produce truly resolution-independent and page-independent output
* Designed from the bottom up for use with outline fonts on the PC

Fonts — *available from Y&Y in Adobe Type 1™ form (ATM compatible)*

* BSR Computer Modern fonts — with accented characters built in
* LaTeX + SLiTeX font set — line, circle, symbol, lcmss*, and logo*
* AMS font set — Euler, math symbol and Cyrillic fonts
* Lucida® Bright + Lucida New Math — a complete alternative to CM

Resolution-independent PostScript files using outline fonts can be printed by any service bureau, not just those with TEXpertise — and that translates into considerable savings for you. Is it perhaps time to get rid of those huge, complex directories full of bitmap fonts?

Y&Y, 106 Indian Hill, Carlisle, MA 01741 — (800) 742-4059 — (508) 371-3286 — Fax: (508) 371-2004

"Lightning Textures works wonders. This has saved me many hours of corrections...I have three books to be completed and this new gadget certainly gives me courage." anonymous graduate student, University of Ottawa

"I find that Lightning sets type on my Quadra 700 faster than the VAX in the Lab can do it. It's a very nice program, indeed." John C. Allred, Los Alamos National Laboratory

"I'll spread the word---Lightning Textures is great!" Prof. Anthony Siegman, Stanford University

"...a fantastic product and I can't imagine going back to an ordinary TEX environment." Chuck Bouldin, Naval Research Laboratory

"I used TEX on a PC for years, but now that I've used Textures I wouldn't (want to) go back to that for anything." George Killough, Hendecagon Corporation

LIGHTNING TEXTURES

"I just typeset 300 pages of two-column heavy math in 7.45 minutes on a Mac IIci -- almost a 4X speedup (actually 3.8). I am very impressed. Good job." Stanley Rabinowitz, Editor of Index to Mathematical Problems

"A few weeks ago, I received a copy of Lightning Textures from you. I was rather skeptical when I ordered it. I could hardly believe that Lightning would work the way you described...But, after testing the program, I am fully satisfied, even more, I am excited about Lightning: Congratulations!" Dr.Bernd Fischer, Universität Hamburg

"I have Textures installed and running -- it's great! Who could prefer other implementations of TEX, I wonder?." Mark Seymour, Springer-Verlag Heidelberg

"It's heaven." Ken Dreyhaupt, Springer-Verlag

"I know that Blue Sky Research did not intend Lighting Textures as a hackers's tool, but it sure can be used that way. It is strange how a simple speedup in the turnaround cycle (of about a factor of 10 to 100) changes your perspective. Lightning Textures is a joy to use." Victor Eijkhout, University of Tennessee

"I am absolutely amazed by Lightning Textures. It beats everything else hands down. I can't imagine what you could be putting in next." Kaveh Bazarghan, Focal Image

503 222 9571 • 800 622 8398 **BLUE SKY RESEARCH** FACSIMILE • 503 222 1643

TUGBOAT

Volume 13, Number 3 / October 1992
1992 TUG Conference Proceedings

Introduction 251 Malcolm Clark / *President's introduction*

Keynote Address 253 Malcolm Clark / *Portable graphics in TEX*

Software 261 Bart Childs / *Literate programming, a practitioner's view*
269 Steve Hampson and Barry Smith / *A high performance TEX for the Motorola 68000 processor family*

Front Ends 272 Harry L. Baldwin, Jr. / *Using a high-level language as an aid in writing TEX documents*
281 Larry F. Bennett / *T-EDIT, a collection of editing macros for TEX*
→ 291 Robert McGaffey / *Automatic tables using* SGML, C, *and TEX*
295 Anthony J. Starks / *Dotex—integrating TEX into the X-window system*
304 Kresten Krab Thorup / *Gnu emacs as a front end to LATEX*
309 Walter van der Laan and Johannes Braams / *Writing reports with more than a hundred people*

Graphics 315 Jackie Damrau / *Discovering graphics in LATEX documents*
322 Robert L. Harris / *Preparing halftones for use in TEX*
327 David Salomon / *Creating shaded rectangles with PostScript*
330 Neil A. Weiss / *Creation and incorporation of PostScript graphics with TEX-formatted labels into TEX documents*

Macros 335 Timo Knuutila / *How to combine multiple languages, PostScript, and LATEX*
341 Victor Eijkhout / *Just give me a lollipop (it makes my heart go giddy-up)*
347 James L. Hafner / *FoilTEX, a LATEX-like system for typesetting foils*
357 Peter Abbott / *Typesetting a magazine the easy way*

File Management 362 Mimi Burbank and Donna Burnette / *Using TEX for a publications database*

Future Issues 372 T. V. Raman / *An audio view of (LA)TEX documents*
380 Dennis S. Arnon, Isabelle Attali, and Paul Franchi-Zannettacci / *Model-based conversions of LATEX documents*

Reports 390 Chris Rowley / *LATEX3 update*
391 Workshops

Participants 393 Participants at the 1992 TUG Meeting

Announcements 395 The Donald E. Knuth Scholarship for 1992 and 1993
396 Calendar
398 TUG 1993 annual meeting, Aston, UK
399 TUG 1993 course schedule

TUG Business 400 Institutional members

Advertisements 402 Consultants
411 Index of advertisers