Editorial Overview

Vancouver in August

This year's annual meeting (my first since 1995 in Florida) had the papers, the attendees, the locale, and the weather all vying for my attention. Every day. It was a fantastic site: that West Coast mournfulness that comes of rain on huge fir trees for the opening few days and then brilliant sunshine for the rest of the week, making the color contrasts outside more like a paintbox than a conference site!

And then there were the papers! Oh yes. Alphabet-soup time, folks. We have more `\acro`s than ever, it seems: XML, MathML, EPS, *DVIPDF*, PDF, HTML, CD-ROM ... The editor's supposed to find the first instance of use in a paper and insist that a definition be provided for those items which are new or very jargon-ish. I tried ... Fortunately, most authors already took care of that step; and where they didn't, well, if the paper wasn't the first one to mention it, then I left it. Because I'm going to assume that not only will you be reading all these papers but you'll also be reading them in order! Of course!

Well ... ok, then ... how about reading all the abstracts at least? After all, that's why we put 'em into all these proceedings articles — sometimes it's the only way for those of you who couldn't be there to get a handle on just where things stand *vis-à-vis* new developments and interesting projects that often seem to wait till the meeting to be unveiled. Granted, that's not where you'll find all the acronyms defined but it's a start.

But before you plunge into the abstracts — or the full-blown articles — stay a moment and find out *what else* went on in Vancouver. Oh my ... we did get up to some fun and games. And I have a suspicion that next year's meeting, given some of its organisers, will yield even more surprises and enjoyment for all in attendance. So, if you can, save up your money, your travel points, your vacation days, and plan on being in Oxford next year in mid-August: the 13th to the 16th. It'll be great! For more details, see p.154 or visit the website: `tug2000.tug.org` ... not "tug00" `;-))`

1 Fun and games

As many of you know from visiting the pre-conference website, submissions for a Poetry Contest were being solicited already long before the start of the conference. However, by the time Wednesday morning of August the 18th had rolled around, the number of submissions was still rather ... euh ... low. So Sebastian stood up and exhorted us, challenged us (while we were between speakers) to do something about it.

It took about half an hour but then there was a steady stream of quiet little walks to Sebastian's end of the hall, a quick smile, a piece of paper changed hands ... and then the speaker's eyes could return to the matter at hand. And thus a Poetry Reading in the Rose Garden was organised for that evening, and the results were unparalleled! You may have heard tales of "inspired" readings — it's all true! Unfortunately, all we can reproduce here are the words. The website has all the poems, all the pictures, and all the prizes.

There was further mirth and enjoyment at doing non-TEX things on the evening of the Banquet, when door prizes were awarded via an apparently random picking of name tags out of a hat. But — how 'random' is it to see most of these going to the same table ... where many of the LATEX3 team were sitting, in fact? I ask you ...

Then it was time to select the Best Paper. The job was pushed onto me — so I promptly turned around to ask six other people for their suggestions. Results? A tie! It was with great pleasure that I could annouce that Don Story and Jean-luc Doumont were the joint winners; judging from the applause, this was the overwhelming view of the audience as well. Congratulations, gentlemen.

2 Recognition

Recognition in a volunteer community can sometimes take a long time to come around. It's not that the awareness or the gratitude aren't there — they are, in spades — but *formal* recognition is something that requires an occasion, a gathering, and, of course, someone to instigate it. How this latter came about for the Vancouver meeting is someone else's story to tell; what I *can* tell you is that Barbara Beeton's 20 years on TUG's board of directors (a steering committee in the early days) and Don Arseneau's many years of support to the entire TEX comunity via packages and answers on `comp.text.tex` are indeed in the category of long-standing contributions that were formally recognised this past August.

Tuesday afternoon saw Michael Downes of the AMS speak of Don Arseneau's place in our directories and our code, and then he presented the man from TRIUMF with a specially commissioned drawing by Duane Bibby. The website has more: Michael's tribute, a list of Don's packages, and Duane's drawing. Visit it — and then count how many packages *you've* used over the years.

On Wednesday evening, at the end of the Poetry Readings, Mimi Jett, TUG's President, had the pleasure of bringing our attention to the fact that Barbara Beeton, in yet another of her many roles, had now been a board member and board meeting attendee for 20 years. A bottle of vodka (courtesy of Irina Makhovaya of MIR Publishers) was probably the best possible choice to celebrate this fact — strong drink for strong work! Pictures and details at the website pages.

Ohhhh, Barbara ... You may want to look at p. 313 ... vodka on paper just won't work, so I have something else for you — and our readers — to remember this 20th year of board-dom.

3 The editorial thing

It's been quite a while since I've been editor. This year's crop of authors and papers has been a true pleasure to harvest. Not that they've sat back and taken everything I've dished out ... but that they've been very good to work with, in order to end up with text both they and I can live with. I'm far more intrusive an editor than most (certainly more than Barbara — but that's also a function of the time that she simply does not have available to her) and yet the authors have been very kind in both accepting — and politely declining — editorial changes.

It's only by the fifth or sixth article that I really hit my stride — which means I have to go back and redo the earlier papers. That's normal. Then come the last few papers and it's almost too much to finish ... and I didn't. Six papers had to wait till after the conference for my editorial sledge and in the interim, yet more second thoughts and doubts about editorial stances which had changed — and then needed to be carried back into previously edited files. So it's been a year, more or less, since Anita's invitation to be proceedings editor; I don't think that's normal — too long — but it's a lot of pages, I tell myself. Perhaps a team approach for the actual editing wouldn't be a bad idea on projects of this size.

3.1 Reviewers

Just as regular submissions to *TUGboat* are refereed, a review process for conference papers also exists. This is crucial when the editor is only aware of a small fraction of what's going on in the TEX community — and knows even less of the details: *I* need this review process even more than the authors! The reviewers were thus able to bring additional information and other possibilities to the authors, who in turn made very good use of the suggestions. I was also fortunate in that only one or two authors had to be asked to review another presenter's paper; 21 reviewers delivered comments on 23 papers. The authors' work you will find further on in these proceedings, the reviewers' names you will find here:

> Don Arseneau, Thierry Bouche, David Carlisle, Don DeLand, Michael Doob, Victor Eijkhout, Robin Fairbairns, Jeremy Gibbons, Denis Girou, Michel Goossens, Eitan Gurari, Kathy Hargreaves, Neal Holtz, Dag Langmyhr, Pierre MacKay, Bernd Raichle, Peter Schmitt, Michael Sofka, Phil Taylor, Ron Whitney, and Mark Wicks.

I would like to thank each of you for having been so generous in offering your time, thoughts, and energies to yet another volunteer activity for *TUGboat* and for the TEX community. I am very grateful for all the added assistance in treating these papers.

3.2 New trademarks

At meetings such as this, there are always new items being presented, some of which are trademarked. Here are two new trademarks to note:

> W3C (World Wide Web Consortium)
> MathType

For regular issues, the list of trademarks can be found on the verso of the title page. Proceedings issues have other stuff on that page, so I reproduce here what we normally cover:

> 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:
>
> MS/DOS is a trademark of Microsoft Corp.
>
> METAFONT is a trademark of Addison-Wesley Inc.
>
> PCTEX is a registered trademark of Personal TEX, Inc.
>
> PostScript is a trademark of Adobe Systems, Inc.
>
> techexplorer is a trademark of IBM Research.
>
> TEX and $\mathcal{AMS}$-TEX are trademarks of the American Mathematical Society.
>
> *Textures* is a trademark of Blue Sky Research.
>
> UNIX is a registered trademark of X/Open Co. Ltd.

4 The organising thing

Conferences always have organisers — yet more volunteers! This year co-chairs Stephanie Hogue and Anita Hoover, along with their committee members, were responsible for the program, local arrangements (for which we also thank the University of British

Columbia Conference Centre's Sarah Johnston), preprint printing, courses, vendor participation, and publicity:

> Don DeLand, Sue DeMeritt, Wendy McKay, Patricia Monohon, Cheryl Ponchin, and Heidi Sestrich.

4.1 Corporate participation

Vendors were active in several ways: display tables outside the meeting room, demonstrations during the meeting, donations of products as door prizes; some also gave papers that yielded detailed introductions and descriptions of some of the new software that's out there. We therefore thank the following vendors for their participation at TUG'99:

> Birkhäuser-Boston, Blue Sky Research (Warren Leach), 4TEX CD (Eric Frambach), IBM (Mimi Jett), Kinch Computing (Richard Kinch), MIR Publishing (Irina Makhovaya), Personal TEX Inc. (courtesy of Anita Hoover), Springer-Verlag, Y&Y, and Zephyr software (courtesy of Ross Moore).

I hope this list is complete!

5 TUG99 Website: `.../tug99/bulletin/...`

It's been just over a year since Anita Hoover, co-chair of the conference committee and technical editor of these proceedings, asked if I'd like to be proceedings editor for TUG'99 — she'd make sure all files were present, process them and confirm that they did indeed run; I would take care of the contents. It took several months before abstracts started coming my way for editing before being posted to the Web. A few months after that and the papers began arriving but it wasn't too hard to keep up. I'd done this before, I knew the routine: Set up my charts, cajole reviewers into lending us some of their time and knowledge, edit hardcopies, input the edits, exchange the files with authors a few times ... and occasionally a few more times ... the usual stuff.

Then something new: Ross offered to convert his paper into PDF format as a demonstration of what TEX on the Web was all about. Sounded fine. Except, somehow that one paper became all the papers; and that job became one that "grew like Topsy", as they say. And the site has continued to grow after the conference, as you all know from the e-mailed notice about the Bulletin being available; initially password-protected in order to allow TUG members first access, the site is now open to all.

And like Topsy, the initial request for a few lines on how the "webification" work had gone became a full-fledged article; we'll have to put it into the next issue of *TUGboat*![1]

However, I would be remiss if I didn't talk a bit about the work that Ross Moore and Wendy McKay have done both pre- and post-conference — it's gone far beyond anything I'd have thought possible.

The website became, in a sense, a demonstration of what many conference papers would be talking about: the migration and integration of TEX source files onto the Web, retaining as much as possible of the original typography and reducing recoding as much as possible. The entire operation was rendered all the more difficult as none of us had planned for it — the source files had therefore to be treated individually, carefully examined for author-specific macros, conversions to LATEX made (it was all being done using `latex2html`), and then after the conference, Ross went back into the files to insert links to Web references which had been cited.

As of this writing (December 28, 1999), the TUG'99 website has had 241 visitors — now surely we can do better than that in showing some interest in Topsy! Visit the site and then, for those of you who are editors, blanch at the prospect of having to deal with a whole new medium of presentation! Me, I've recovered from blanching ... but let next year's editor(s) be warned `;-))`

As mentioned, Ross has written up a blow-by-blow account of the webification process, which we will carry in a future issue of *TUGboat*. His trials and tribulations are those of any first-time effort: things can only go better next time around! While it is perhaps obvious to those of you who have already been involved in paper/web publishing, preplanning is essential to keep everyone's sanity within the bounds of 'normal'. I feel almost like I was being left behind, in some ways, as not only the papers, but Ross and Wendy's work, was moving well beyond my sphere of activity and experience. It's exciting — and I've had some novel uses added to my repertoire these past weeks as Mimi Burbank (our long-suffering production manager) and I have worked to check all the little bits and pieces of text and items that seem to come to mind only at the very end — and it's a bit intimidating. I'll have to spend some time thinking about just how far into the "web-thing" I want to venture ... but it's been a heck of an experience, I can tell you!

[1] Note that all papers at the website are the preprint versions; the electronic versions of the papers found in these *TUGboat* pages will be slightly different, and of course several papers will have a "Post-Conference Update".

5.1 Map of TUG meetings and members

Something else that Ross and Wendy worked on is a massive map showing where all the TUG'99 attendees had come from! The map is best viewed on the web — it just doesn't come across as well in print. It also shows where all past meetings have taken place (see also p. 314). So if you want to see a demonstration of some of the techniques described in their paper, go to `/preprintmap/node2.html` in the Bulletin webpages. The first image is a `.gif` file but if you have the capacity, download the `.pdf` version (833K).

5.2 The 'TEX Friendly Zone'

One more thing (their energy is boundless!) Wendy has also put up a "TEX Friendly Zone" logo, a nifty new Bibby drawing that's available for download and printing. Here's what it's all about:

> The TEX Friendly Zone (TFZ) logo is another one of Duane Bibby's[2] beautiful drawings of the TEX lion.
>
> Permission is granted to print the TFZ image and post it in your office or any place where TEX Friendly people work!
>
> You are also encouraged to get new members to join TUG and enjoy the camaraderie of TUGees who willingly share their knowledge and expertise to make the typesetting world a better place!

5.3 Photos and Images

And then there are the photos. We have to thank Warren Leach of Blue Sky for such fantastic images, both still and in QuickTime video, and also Kirin Bahm and Jean-luc Doumont. Digital cameras are what yielded the wonderful results on-screen. On the other hand, for print, scanned pictures from a good still camera are still a better route (it's all a question of image resolution).

My efforts, on the other hand, with an old defective camera, were less than stellar ... I'd highly recommend that next year's organisers enlist the aid of digital-camera owners to capture images if there are any plans for web publication of pictures. Again, these images are all at the website. There are lots more photos (and in color, of course). Information on how the photos were taken is also available.

The full story on the Grimm Exhibition is also available from the website, along with some of the posters (again, in glorious color). If you have the bandwidth and the space, try to look at the `.pdf` versions of these posters — it's a wonderful show just watching them 'develop', layering the colors and the shapes ...

6 Production notes

There are actually two "productions" involved in creating this year's proceedings: the usual one for print copy, and the first-time (for us) of e-versions of the preprints.

Abstracts were solicited last fall, and, after I'd edited them, the texts were posted to the website — the "webbing" began early on.

To help with their articles, macros and user guides were made available to authors (again, from the TUG website). As the work progressed, I made notes on both my hardcopies of the guides on points where we might improve or add things, particularly to the guide for plain TEX.

We had 23 papers submitted, 7 in plain TEX, 16 in LATEX; two papers were not submitted by the deadline, and a third was withdrawn. Both flavours of TEX presented me with plenty of new coding styles, package options, and just really good examples of how to do things. As well, a good number of papers used BIBTEX, something which I learned to use — and then control ... a little! In short, editing our proceedings may start with the words but it often progresses deep into TEX itself — lots to learn in very short order.

Authors dropped off their files at a designated ftp site, and then Anita Hoover, the technical editor, tested them at her site to ensure all files had been supplied before moving them to SCRI.[3]

I approached potential reviewers to read and comment on each paper, and as mentioned earlier, these people did a terrific job. I appreciated their willingness to help, and I believe the authors appreciated finding a willing and knowledgeable audience to check their work.

Once authors had addressed reviewer issues, their revised files were again processed to ensure they still ran, and Anita then generated `.ps` files for me to pick up, print here, and begin editing.[4] I

[2] "This year the familiar and much loved TEX Lion was brought back to enhance and illustrate our activities in a unique way. Duane Bibby produced several wonderful TEX Lion art works for the TEXLive4 CD, TUG'99, TUG2000, the Special Recognition Award, and the TFZ logo. We are happy to have found him again, and we thank him very much for doing these very special drawings — as only he can — at quite short notice." [footnote in original – Ed.]

[3] The Supercomputer Computations Research Institute at Florida State University, where all *TUGboat* production takes place.

[4] As SCRI has to be complete in its collection of fonts, packages, and utilities, there's no point to my duplicating that environment at my site. Eventually, though, as the editing cycle progressed, I did process the bulk of the files here in order to deal with final line-, column-, and page-breaks.

prefer to edit first on hardcopy, mainly so I can see all the pages at once, and also backtrack now and then `;-))` before changing the files, which were processed as necessary during this time to ensure that edits weren't generating errors. Then began the exchange of files with the authors: the editorial dance, as it were. Again, all went very smoothly, and so here we have a largish issue of *TUGboat* which represents the written record of most of what went on at this year's annual meeting in Vancouver, supplemented this time by a good selection of images and additional texts on the conference website. That is, what we didn't have room for, or what is not suitable for print (i.e., images, especially color ones), you'll find up on the Web.

I hope you enjoy reading — whether on paper or on screen — about the work that's currently engaging the TEX community as it does indeed work its way onto the Web, tangled and tortured though that route may seem at times. And throughout, I think you will find a renewed and invigorated sense of being on the forefront of developments that affect not just the math community but all of us. Recycling source files after our hard-won battles with the code, and seeing them emerge on the other side, perhaps even in colour (if we've done that pre-planning!), means our work is going to enjoy a very long life no matter what the medium.

6.1 Some final thank-yous

All of the above efforts are for naught unless they make it to paper. And all the paper copies are for naught unless they're properly accompanied by the easily forgotten elements to an issue: the calendar, the titlepage, the covers, the table of contents, and the final sequencing and assignment of page numbers to *all* the elements.

Ensuring that we don't forget any files, we have Barbara, who checks and calmly corrects what has already been checked, and who has seen to it that this editorial has also been checked and edited.

Ensuring that we don't forget any printed pages, we have Mimi Burbank, who has looked at all my nicely formatted final files and then made suggestions for improvements to many, has seen to the inclusion of poems throughout the issue, and then has stood by the printer, in sickness and in health (the printer's, I mean!), to ensure that the pages are all there and that the whole package is shipped off to the printer (the company, I mean!).

As I said near the beginning, recognition requires an occasion and someone to instigate it. That is as true for the launching of a conference effort as it is for safe delivery of its printed proceedings. And so I must close by thanking these two women for their work in seeing these pages through their final days and hours of production, after the editorial work had been completed. Thank you. As always.

7 Where to find things on the TUG'99 website

Start off at `tug.org/tug99/bulletin`, the TUG'99 Post-Conference Bulletin, and read the warnings about this being a photo-intensive and colorful site. Scroll down to `Click here ...` and take a coffee along for the trip. Color monitor recommended. Also, please note that the password restriction (in place to allow TUG members first dibs) has now been lifted.

introduction	`/frontpage`
papers (index)	`/preprintmap/node1.html`
poetry	`/poems`
awards	`/award`
	`/otherawards.html`
The Apocalypse	`/grimm`
photos (index)	`/photoalbum`
world map	`/preprintmap/node2.html`
TFZ logo	`/tfz`
TUG2000	`tug2000.tug.org`

And one final place to visit:

`/photoalbum/node15.html`

A real cutie!

◇ Christina Thiele
15 Wiltshire Circle
Nepean K2J 4K9, Ontario Canada
`cthiele@ccs.carleton.ca`

Nevertheless, the grunt work is done at SCRI, and for that we are always grateful to Mimi and the people at that site.

TUG '99 Program
TEX Online — Untangling the Web and TEX

Monday, August 16, 1999
TEX and Math on the Web

Stephen A. Fulling
"TEX and the Web in the Higher Education of the Future: Dreams and Difficulties"

Patrick D.F. Ion
"MathML: A Key to Math on the Web"

Doug Lovell
"TEXML: Typesetting XML with TEX"

— Break —

Paul R. Topping
"Using MathType to Create TEX and MathML Equations"

Workshop: "LATEX to XML/MathML", *Eitan Gurari and Sebastian Rahtz*

— Lunch —

Chris Rowley
"Models and Languages for Formatted Documents"

D.P. Story
"AcroTEX: Acrobat and TEX Team Up"

— Break —

Panel Discussion: "TEX and Math on the Web", *Stephen A. Fulling* (moderator)

Workshop: "Using LATEX to Create Quality Interactive PDF Documents for the WWW", *D.P. Story*

Tuesday, August 17, 1999
Customizing Document Layout

Jean-luc Doumont
"Doing it my way: A Lone TEXer in the Real World"

Peter Flynn
"The vulcan Package: A Repair Patch for LATEX"

Vendor Presentations

— Break —

David Carlisle, Frank Mittelbach and Chris Rowley
"New Interfaces for LATEX Class Design: Parts I and II"

— Lunch —

Workshop: "Writing Class Files: First Steps", *Michael Doob*

— Break —

Workshop: "Converting a LATEX 2.09 Style to a LATEX 2_ε Class", *Anita Z. Hoover*

TUG Business Meeting

Wednesday, August 18, 1999
TEX in Publishing

Kaveh Bazargan
"Multi-Use Documents: The Role of the Publisher"

Frederick H. Bartlett
"Very like a nail: Typesetting SGML with TEX"

Harry Payne
"Making a Book from Contributed Papers: Print and Web Versions"

— Break —

Robert L. Kruse
"Managing Large Projects with PreTEX: A Preprocessor for TEX"

Arthur Ogawa
"Database Publishing with JAVA and TEX"

Paul A. Mailhot
"Implementing Dynamic Cross-Referencing and PDF with PreTEX"

— Lunch —

Hu Wang
"A Web-Based Submission System for Meeting Abstracts"

Petr Sojka
"Hyphenation on Demand"

Jonathan Fine
"Active TEX and the DOT Input Syntax"

— Break —

Panel Discussion: "TEX in Publishing", *Kaveh Bazargan* (moderator)

Thursday, August 19, 1999
Fonts, Graphics, and New Developments

Jean-luc Doumont
"Drawing Effective (and Beautiful) Graphs with TEX"

Wendy McKay and Ross Moore
"Convenient Labelling of Graphics, the `WARMreader` Way"

— Break —

Sergey Lesenko and Laurent Siebenmann
"Viewing DVI Files with Acrobat Reader — DVIPDF Gives Birth to AcroDVI"

Alan Hoenig
"*MathKit*: Alternatives to Computer Modern Mathematics"

Vendor Presentations

— Lunch —

F. Popineau
"fpTEX: A teTEX-Based Distribution for Windows"

Jeffrey McArthur
"Managing TEX Software Development Projects"

Timothy Murphy
"JAVA and TEX"

— Break —

Panel Discussion: "The Future of LATEX", *Arthur Ogawa* (moderator)

KEYNOTE:

TEX and the Web in the Higher Education of the Future: Dreams and Difficulties

Stephen A. Fulling
Mathematics Department,
Texas A&M University,
College Station, Texas,
77843-3368 USA
fulling@math.tamu.edu

Abstract

New technology provides an opportunity to move mathematical and technical education out of the lecture-homework-test mold into modes that do more to develop students' communication skills, teamwork, attention to quality, and overall responsible, mature behavior. In practice, however, severe and presumably unnecessary obstacles are encountered, mostly connected with the difficulty of transmitting mathematical notation electronically.

As I document from personal experience, these obstacles are bad enough for an instructor providing information to students over the Web, and worse when students themselves are expected to communicate mathematically. I describe a partially successful scheme for carrying out peer review of homework papers over e-mail and the Web, resulting in a student-written solutions manual, and I offer a wish list of technical improvements.

In memory of Norman Naugle and James Boone

There are two kinds of people, classified by their reactions to new technology. People of the first type say, "Now we can do new things — things that were impossible before." The other people say, "Now, finally, for the first time, we can do the old things *right*!" Tim Berners-Lee, who made this conference possible and necessary by creating the World Wide Web, surely belongs to the first camp. Donald Knuth, with his injunction to go forth and create beautiful books, is perhaps the most successful exemplar of the second mode of thinking.

The advance of computer technology is having profound effects — actual or projected — on undergraduate education. Most of the discussion takes place in the first mode: laboratories with computer algebra systems, pedagogical Java applets, distance education to reach new audiences, greater attention to numerical methods in the content of courses, etc. Personally, I feel more capable of contributing in the second mode, using the computer, the Internet, and the Web, mainly as marvelous communication tools that address the discontents of modern mass higher education. Here I shall describe two efforts in this direction, with emphasis on the frustrations I encountered because our two most marvelous tools, TEX and the World Wide Web, do not communicate well with each other.

I am not a computer professional. I'm a university teacher of mathematics, trying to make significant contributions to that art while striving to keep a research career alive. This gives me little time to write my own software, and not even enough time or facilities to find and evaluate all the software that already exists. It is a great honor and responsibility to be scheduled as the leadoff speaker of this Annual Meeting. The professionals should regard me as something like Dilbert's Boss (Adams, 1996, for example): I will present you with a list of technical demands; some of them may be ridiculously impractical; the rest are those you must implement to keep academic customers happy in the next decade. Your job is to distinguish between the two categories and to take action on the good half. I am making an effort not to dwell on any existing partial solutions that I may be aware of, because I know that my information is incomplete; it is up to the providers of those solutions to speak for themselves.

What I want for myself yesterday (or at least tomorrow)

Let's start with the obstacles facing an instructor who wishes to make text material with significant mathematical content available to students over the Web.

For two academic years (from 1996 to 1998) I was one of two calculus teachers in an experimental freshman engineering program integrating calculus, physics, English, chemistry, and two introductory engineering courses into a unified curriculum. (This is a product of the Foundation Coalition, a consortium of universities and colleges supported by the National Science Foundation.) To meet the demands of a complete physics sequence in the freshman year, it was necessary to revise the traditional calculus syllabus by moving large quantities of multivariable material from the third semester into the first two. No textbook on the market does that! Therefore, I wrote some textbook fragments covering vectors, multiple integrals, Taylor series, and a few other topics, at the level and from the point of view needed. (See the URLs listed at the end of this article.)

As a long-time user of TEX, my first inclination was to write everything in TEX and post the resulting Masterpieces of the Publishing Art onto the Web. But that smelled too much of using the computer monitor as a bulky imitation of a printed page. In particular, hypertext features would not be available. Also, the screens would not have the look and feel to which my Generation X audience is accustomed — possibly a serious tactical error in winning their confidence and engaging their enthusiasm. I went to the opposite extreme: I wrote the documents in HTML, in the hypertext and bulleted-list style. Whenever possible, the mathematical expressions were written in HTML. Whenever I just couldn't stomach the results, or a passage was particularly heavy in formulas, I wrote a patch of TEX and linked the `.dvi` file to the higher-level HTML page. (Since not every browser the students were likely to use was configured to handle `.dvi` files, PDF versions had to be supplied as well.) The results still have a somewhat amateurish appearance because the TEX (and the graphics) are not properly integrated into the HTML pages. (With considerably more work and self-education I could have done better, but I did not have the time or the expertise.)

But the world shouldn't be like this!

1. NON-NEGOTIABLE DEMAND: It must be possible to place all standard mathematical symbols on an HTML page, including the Greek letters, integral signs, square roots, built-up fractions, etc., with confidence that they will be displayed properly by every standard browser. ("Standard" may need to be redefined but the new standard must be implemented!)
2. The screen appearance of the mathematical expressions must be of "typeset quality". NOTE: this does not necessarily mandate Computer Modern fonts. Indeed, one of the obstacles to widespread acceptance of TEX on the Web is that the CM fonts were designed for high-resolution devices, and the more slanted ones, especially, are barely readable on screen except at very large magnifications. Some compromise in the design of the math fonts may be necessary and acceptable.

I think that anyone who has tried to type mathematics in HTML, even when all the necessary characters and features (such as superscripts and subscripts) are available — or who has looked at the syntax of MathML — will agree with the following requirement:

3. It must be possible for the author to create mathematical expressions by typing raw math-mode TEX code into the HTML (or XML) file (inside some SGML-type wrapper, of course).

You'll note that what I'm calling for here is "encapsulated TEX" (Murphy, 1991). Indeed, for many purposes I am perfectly happy with the *text* of an HTML document as browsers display it on the screen and PostScript printers print it out. It is the mathematical expressions that are unacceptable at present. Moreover, the new MathML language in its raw form is not writable or readable by human beings — it would be like writing a text document directly in PostScript. A human-oriented language is needed for input to MathML. But we have such a language — it is TEX! The problem is to get the rest of the world to recognize it as the preexisting standard that it is.

This leads back to a question that I'm sure many of you wanted to ask earlier: Why don't I use a program like `latex2html`? In part, that is a personal prejudice; my natural language is `plain`, with an accumulation of personal macros, and I would need to translate my files into LATEX to use such programs. But I refuse to make that effort because, in any event, I can't accept the results as a permanent solution:

4. The mathematical expressions must resize properly when the user changes the font size in the browser.

5. The mathematics must reside in the HTML file, not in dozens of auxiliary files.
6. The system must not rely on an engine located on a remote third-party machine. (This seems to rule out MINSE, for example.)

So, that is what I want for myself. And I want it yesterday — because the curriculum pilot project is now over. As usually happens, something much less ambitious was mainstreamed by my institution; the course materials produced for the experimental classes remain on the hard disk, as a monument to ... something.

What I want for my students today (or at least next year)

I frequently teach courses for engineering and science majors in their last two undergraduate years, covering topics in mathematics such as linear algebra and Fourier series. I have concluded that, even at this advanced level, our standard teaching practices do very little to develop habits of mature, independent, professional behavior. This is not the place for a harangue on pedagogical philosophy. (See references under "Homework review" below.) Suffice it to say that we stand at a crossroads: *Educators can use computers to restore literacy, or to drive the last nails into its coffin. Freedom from paper* is not the same thing as *the death of the alphabet.*

One of my attempts to address this situation is a drastic change in the handling of homework assignments. Instead of requiring every student to turn in every exercise on the list, I assign each student only one or two problems per week. But these problems must be taken seriously: The instruction is to produce *a carefully written, complete, formal solution, comparable to a worked-out example in a textbook.* The class, collectively, solves most of the problems, thereby producing a "solutions manual". This is a class project, their legacy to the students of future semesters. All the papers turned in on one exercise are given to a team of two or three students for peer review. Subject to quality control by the paid grader and myself, they write a brief critique of each paper and choose the best one for "publication". The criteria for evaluating papers include presentation as well as correct content; in descending order of importance, they are:

- mathematical correctness
- completeness
- organization
- pedagogical effectiveness
- sentence structure and punctuation
- brevity and style
- grammar and spelling
- neatness

After several years of working purely with paper, this system moved to e-mail in 1996. I expected that keeping the reviewers' reports electronic would cause a huge increase in efficiency, and for me it did; however, I soon faced a revolt from the students, who said they were spending too much time on keyboard busywork instead of learning math, and from the grader, who found the chaotic e-mail files harder to work with than paper. Then a miracle happened: a student in the class, Justin M. Sadowski, showed up in my office with a `C++` program to automate the mailing of reports. This program operates on the departmental UNIX systems where our students have accounts. A menu guides the student reviewer to type in the account names of the student authors, and the program spawns sessions of the `pico` editor to write the reviews and a summary report to the instructor into template files. At the end it mails the messages to the authors, grader, and me.

Students are strongly urged to make their homework solutions visible on the Web as well as on paper. The long-range goal is an on-line solutions manual, recreated each semester by a new class. But since these are courses in mathematics, not computer usage, the system must run on the students' preexisting computer skills, which vary widely. At present I estimate that about:

- 50% of the papers are still hand-written
- 20% come from a word processor but are not Web-displayed
- 20% appear on-line, either in ASCII text or in HTML, the latter usually converted either from Microsoft Word, or from *Maple* (which all our students learn as freshmen)
- 10% are written in TEX

These divisions are moving up (in the higher-tech direction) every year.

Why don't more students use TEX? Well, one of my students wrote:

> I spend enough time trying to get my programs to compile. I don't want to have to debug my [homework] papers!!

They are not necessarily encouraged by their senior mentors, either. Although TEX has become the indispensable standard in academic mathematics and physics, the situation is different in engineering, where most of my students come from. Two years ago in a discussion at a conference on engineering

education I suggested that we encourage the use of TEX, and the reaction of an engineering professor was, "Surely you're not still using that old thing!" He couldn't believe that mathematicians hadn't all switched to Microsoft Word. The fact that most Web users still can't even read TEX output is another indication that we have a problem:

7. A plug-in or helper application to display `.dvi` files should be standard with every Web browser.

A look at the mathematical documents created by my students and, even more, a look at the reports they have written on their peers' work, shows that serious problems remain. The students face the same obstacles listed earlier in putting mathematical expressions on the Web or into their printed papers, compounded by their inexperience. So, let us extend the list of demands:

8. It must be possible for *students* to produce decent mathematical documents, both in print and on the Web.

Universal participation requires that:

9. The system must be platform-independent. (At the very least, it must exist in both UNIX and Windows versions, which communicate flawlessly with each other.)
10. The cost to students (in required software purchases) must be minimal.
11. The time spent in learning software syntax must be minimized.

These last two problems will be alleviated if the students use the software throughout their undergraduate careers, not just in one course taught by an eccentric professor. Unfortunately, "it is a characteristic of those in the [college teaching] professions to resist change unless it is change that they tried to start."[1]

How can we minimize software learning time? Does it mean giving up TEX? On the contrary, I return to the notion of encapsulated TEX. I believe that most of TEX's reputation for being hard to learn and to use is related to high-level document formatting. Most people first try out TEX on a small document, such as a job résumé or a letter, that would be trivial to type as a pure ASCII file where the typist has direct control over the placement of every character: the beginner soon discovers that heroic measures are necessary to override TEX's default behavior of reformatting paragraphs and discarding all the redundant white space. But all that is really necessary for most purposes is the math mode of TEX.

12. We must create an environment that makes it worthwhile for every student to learn basic TEX math mode syntax. (Learning to create complete TEX documents, however, may be left as the student's option.) Instructional material for TEX beginners should be reorganized to reflect this priority.

TEX math syntax is a very efficient and logical way of typing in mathematical expressions, far superior to raw SGML-type languages. Hunt-and-peck menu systems are arguably easier to learn, but in the long run excruciatingly slow to use. The TEX community simply hasn't succeeded in getting this message out to the rest of the world.

In those fields where TEX has become the standard method of producing serious documents, its basic math syntax ("pidgin-TEX") has also become a rough-and-ready way of incorporating math into e-mail and other informal pure-ASCII communications. This is another major reason why all students should learn this part of TEX. My student reviewers are supposed to provide constructive criticism of their peers' papers and to describe in their top-level reports the most serious and systematic mathematical errors they observed. Very often, what they write is something like "You made a mistake in part (c)," leaving the author and me wondering what the mistake is. A big part of the problem is the difficulty of discussing mathematics with any specificity in an ASCII text message. More generally,

13. All students (from the freshman year on) must be enabled to discuss mathematics easily with their teachers and with fellow students, in e-mail, list servers, chat rooms, etc. Pidgin-TEX should be promoted as a medium for this.

Of course, a more sophisticated system that made it possible for the students to annotate one another's documents with electronic sticky-notes would be even better.

Conclusions

From my standpoint as a university teacher, the overwhelming issue right now is how to present decently typeset mathematics on the Web. Presumably, in the near future this means: (1) tools for turning TEX input into MathML output, and (2)

[1] The original quotation (Childs, et al., 1995, p. 303) refers to the *programming* professions, but one of the authors has assured me that the theorem has broader validity.

assuring universal availability (on college campuses, at least) of browsers that can handle MathML.

Students should be able to create decent mathematical documents of their own and also to discuss mathematics in e-mail, etc. TEX math mode provides a suitable language for the latter if we promote it adequately.

The academic scientific community laid the golden eggs for the exploding computer and software industries. In that light, the lingering difficulty of including mathematical notation in Web presentations and in electronic mail amounts to a scandal. It is up to the TEX Users Group to:

- solicit its membership's contributions, at all technical levels, toward solving this problem
- exert pressure on commercial developers (and on agencies such as the W3 Consortium) to steer developments in directions that meet these needs[2]
- continually keep its membership and the rest of the academic community informed of what options are or will be available
- keep TEX in the mainstream of electronic communications, so that the rest of the world does not come to regard it as a relic of the '80s

Some links in lieu of references

Mathematics in the Foundation Coalition (FC). A detailed report on the FC freshman calculus program at Texas A&M University is at

`http://www.math.tamu.edu/~fulling/fc/`

A summary of its goals and practices appears as

`http://www.math.tamu.edu/~barrow/aseepaper.html`

The Web pages for students to read in place of textbook sections can be reached directly as

`http://www.math.tamu.edu/~fulling/coalweb/`*xxx*`.htm`

where *xxx* = `vectors`, `cenmon`, `diffs`, or `taylor`, for instance; all are accessible from the course syllabus pages within the report.

Homework review. The home pages for my sections of Mathematics 311 are at

`http://calclab.math.tamu.edu/~fulling/m311/`

Instructions for the students are

`http://calclab.math.tamu.edu/~fulling/m311/s99/handout.dvi`

for class procedures, and

`http://calclab.math.tamu.edu/~fulling/m311/s99/instruct.txt`

for operating Sadowski's homework review generator. Samples of the resulting student reports and solutions can be seen at

`http://calclab.math.tamu.edu/~fulling/hwk/311s99/sol2.html`

and similar URLs (all accessible from the course home pages). The rationale for the peer review system is described in

`http://calclab.math.tamu.edu/~fulling/iawpaper.ps`

(written at an early stage when all the communication was still on paper). These ideas were heavily influenced by the collections in Connolly and Vilardi (1989) and Sterrett (1990).

General issues. Primarily to introduce students to TEX and to the problems of mathematical electronic communication, I have set up three sets of links:

`http://calclab.math.tamu.edu/~fulling/`*xxx*`.html`

where *xxx* = `webmath`, `webtex`, and `viewers`.

`webmath` lists on-line references on the problem of putting mathematical notation into Web pages and e-mail, with links to proposed solutions, including MINSE, TTH, *Techexplorer*, and `latex2html`.

`webtex` provides information about TEX, including its derivatives with (allegedly) simpler input interfaces, such as `StarTex` and *Scientific Word*. It has links to CTAN and TUG's home page.

`viewers` deals with viewers for `.dvi` files, which people who do not aspire to write TEX themselves nevertheless need to read TEX documents placed on-line in the most compact and convenient format.

References

Adams, S. *The Dilbert Principle*. Harper–Collins, New York, 1996.

Childs, B., D. Dunn, and W. Lively. Teaching CS/1 courses in a literate manner. *TUGboat*, 16(3), 300–309 (1995).

Connolly, P. and T. Vilardi, eds. *Writing to Learn Mathematics and Science*. Teachers College Press, New York, 1989.

Murphy, T. PostScript, **QuickDraw**, TEX. *TUGboat*, 12(1), 64–65 (1991).

Sterrett, A., ed. *Using Writing to Teach Mathematics* (MAA Notes No. 16). Mathematical Association of America, Washington, 1990.

[2] I reiterate that I have avoided positive remarks on various products because they would inevitably have been incomplete and unfair.

MathML: A Key to Math on the Web

Patrick D.F. Ion
Mathematical Reviews
P. O. Box 8604
Ann Arbor, MI 48107, USA
ion@ams.org

Abstract

As the Web gains in importance and as the needs of mathematical formalism on the Web are beginning to be met by MathML, it is an opportune time to reflect on the design decisions made by the W3C Math Working Group that resulted in the verbose markup language for transport of math on the Web that MathML turns out to be. The TEX community need not be frightened by the advent of MathML but may learn to work in the Web environment it provides.

The presentation will describe some of the key aspects of MathML and explain how it has begun to be used (by August 1999 the support for MathML may be expected to be much greater in a practical sense than it is now). Expected future developments and roles will be commented upon.

Introduction

The Math Working Group[1] of the World Wide Web Consortium[2] certainly believes that MathML, Math Markup Language (Ion and Miner, eds., 1997), is a key to math on the Web. It is a protocol for transferring mathematical knowledge, and for building tools to manipulate it, which shows great promise. There are already several implementations and tools based on MathML. Maybe its greatest strength is that MathML is a specification which is open to view and has been adopted as a Recommendation by the World Wide Web Consortium (W3C).

This contribution will discuss some of the context in which MathML has been developed; some of the design decisions that went into it will be illustrated in the live presentation.

History

In 1897, at the First International Congress of Mathematicians in Zürich (Rudio, 1898) there were not many talks.[3] Striking among their titles is "Über Pasigraphie, ihren gegenwärtigen Zustand und die pasigraphische Bewegung in Italien".[4]

The presenter, Ernst Schröder, an algebraist and logician from Karlsruhe[5] began his talk by saying that if there were any topic that really belonged at an International Conference of Mathematicians, then it was pasigraphy. He was sure that pasigraphy would take its rightful place on the agenda of all succeeding such conferences. Perhaps it is needless to say it did not.

Schröder then went on to disagree with the distinguished chair of the session, G. Peano,[6] by saying that he did not think that Leibniz's problem of providing an *algebra universalis*, a symbolic calculus for mathematics, had been solved. Peano (1858–1932) had just begun to publish, in 1894, his four-volume treatise intended to provide just that. It is here, in fact, that Peano's axioms for the natural numbers are to be found, along with axiomatizations and highly symbolic representations for much of arithmetic, algebra, geometry and calculus.

Schröder offered some of his own considerations on the topic of universal symbolics for math as part of logic. He favored a system, near that of C.S. Peirce (1867), using eighteen special symbols.

It is clear that the progress of science in general, and mathematics in particular, depends on there being a representation of its findings external to

[1] http://www.w3c.org/Math, co-chairs: Angel L. Diaz (1998–2000), Patrick D. F. Ion (1997–2000), Robert L. Miner (1997–1998).

[2] http://www.w3c.org.

[3] I thank R. Keith Dennis for showing me his copy of the proceedings in 1997.

[4] "Pasigraphy, its present state and the pasigraphic movement in Italy" (Schröder, 1898). Pasigraphy is an artificial international language using characters (as mathematical symbols) instead of words to express ideas.

[5] Perhaps best known today from the so-called Schröder-Bernstein theorem.

[6] The Italian mathematician who can be considered leader of the pasigraphic movement in Italy.

individuals. In this way the evolving knowledge can be shared. The standard reference work on the history of mathematical notation is by Florian Cajori (1928/29).

The ontology of mathematics has changed over the centuries, so what the objects of mathematics are is by no means a given. Very roughly speaking, and seen from the present day, for the early Greeks math was geometry. Then came along a revolution started by Descartes who put forward a successful algebraic form of geometry through the introduction of coordinates. Throughout this time what numbers were was really not up for discussion although there were, for instance, differences drawn between rational and irrational numbers, and later between algebraic and transcendental numbers. At the end of the last century and the beginning of this, a new view held that all math had to be founded upon set theory and logic. And now that is seen as not all that satisfactory, so that categories, toposes, or the new developments of mathematical logic are viewed as fundamental.[7]

These remarks have been made because there is an obvious sense in which the W3C Math Working Group is just trying to create a modern form of universal symbolic language. It is not intrinsically an easy task.

The W3C — World Wide Web Consortium

The advantage of MathML mentioned above, that it is a public specification and a recommendation adopted by the W3C, stems from the sort of organisation the W3C is.

The W3C is a consortium of some 340 or so organizational members,[8] mostly commercial entities. They include big computer and software companies — such as IBM, Hewlett-Packard, Sun, Microsoft and Netscape, as well as smaller ones — and others from, for instance, the big aircraft or electricity industries — Boeing and Electricité de France, to name two. In general, the W3C is an organization joined by those who wish to understand and influence the development of the protocols and mechanisms that control the functioning of the Web. The W3C Director is Tim Berners-Lee, generally recognized as the inventor of the Web. The W3C is fully international in scope, with main offices in Cambridge, Massachusetts, USA (MITLCS), in Grenoble, France (INRIA) and in Tokyo, Japan (Keio U); it receives additional support from both DARPA and the European Commission.

[7] See the potted history in, say, a book by Godement (1998).

[8] `http://www.w3.org/Consortium/Member/List`

The W3C forms working groups (WG) to study and produce recommendations on subjects concerning the Web. These range from HTML, HyperText Markup Language (Raggett and Jacobs, eds., 1998), the basic markup language for the Web, to PICS, the Protocol for Internet Content Specification (World Wide Web Consortium, 1997), which allows people to control access to pages based on content ratings. A list of all the concerns of the W3C is available at their main Web site mentioned above.

Particularly relevant to the present subject are the working groups centered around XML, an acronym from eXtensible Markup Language (Bray et al., 1998) and XSL/CSS (eXtensible Style Language and Cascading Style Sheets (Deach, 1999; Bos and Lie, eds., 1999; Bos et al., 1998).

Members of the W3C may request representation on any WG of interest to them (one representative and possibly one alternate at most on a WG, together entitled to one vote in deliberations) and there may be invited experts from outside the W3C in a WG, as deemed appropriate. The W3C tries to work by consensus as much as possible. The limitations on voting are there so that problems will be sorted out on technical merit rather than being subject to gross commercial pressures. There are guidelines and procedures, including voting if necessary, developed by the WG that deals with that aspect of the W3C.

SGML, HTML and XML

The specification for which the W3C is best known is HTML, presently at version 4.0. This was introduced by Berners-Lee along with the linking and transfer protocols of HTTP, Hypertext Transfer protocol (Fielding et al., 1998). In the beginning this was a language developed for a specific purpose, and not necessarily consonant with any other standards. However, it was realised that a few changes to HTML would make it a markup language obeying the principles of SGML, Standard Generalized Markup Language (International Standards Organization, 1986; van Herwijnen, 1994). SGML is an ISO international standard that describes a way of writing down document markup. It provides a very general framework, both verbose and complicated to use. There has been a lobby for some years trying to push SGML standardization as a highly desirable means for publishing production, which facilitates document re-use amongst other things.

However, the difficulties in using SGML standards in practice meant that it was unthinkable that those standards be taken straight over to the Web. True, SGML adoption would have provided a great

wealth of possibilities for Web publishing. But a Web browser would essentially incorporate a full SGML parser and, worse still, it would have to deal with the DSSSL, Document Style Semantics and Specification Language (10179:1996(E), 1996) for actual display.

To use SGML strictly, every document should be parsed against an explicit DTD (Document Type Definition) to be sure that it is a correct instance of some general class of documents and markup. This would require a level of rigor in SGML Web page programming in clear conflict with the use, and usefulness, of the Web. An SGML Web, it was feared, would rapidly age. Pages might not be produced because it would be too difficult to provide the level of correctness required for complex SGML browsers to function; also, standards would be difficult to observe in writing pages and in writing software, and the increase of entropy toward a chaotic breakdown of the Web would be rapid. Thus a cut-down, or simplified, version of SGML began to be developed for use with the World Wide Web, under the aegis of the W3C.

XML is a restricted form of SGML as a specification for markup languages. The most obvious thing unchanged is the tagging structure. As in SGML, elements of the document are marked up with tags, to form phrases like `<footag>`*element content*`<footag/>`. An element need not have any explicit content, in which case it is of the form `<bartag />`. There is a simple syntactic difference from SGML: XML elements which are not containers have tags ending with the tokens `/>`. Then again, in XML, markup has to be complete; it is not allowed, as it can be in an SGML markup specification, to shorten the markup by leaving out those end tags which can be inferred as necessary because some overall containing element has finished. For instance, paragraphs have to ended explicitly with a `</p>` and cannot be assumed to have done so just because another one has started with a `<p>`.

The next important matter for the Web is that the language design is intended to allow the processing, at least to some reasonable level, of pages without a declared DTD. An XML document can be well formed without being an instance of some DTD, although being well formed essentially means it would be possible to construct a DTD which has the document as a valid instance. So a browser which can parse XML can be allowed to process a free-form page, if XML syntax is respected.

There are many other technical differences from SGML which allow XML parsers to be much easier to. Many simplifications have come out of years of experience with SGML.

Again of great importance, perhaps more than its technical quality, is that XML is a Recommendation of the W3C. The corporate members of W3C will support its place on the Web. Browsers are being made with with XML support (e.g., by both Microsoft and Netscape). Tools, such as editors, are being produced to enable XML document creation. Associated specifications such as XSL and CSS for formatting (for an overview, see W3C Style, 1999), a DOM, Document Object Model (Apparao et al., 1998) for the Web, the RDF, Resource Description Framework (Lassila and Swick, eds., 1999) model for providing metadata,[9] as well as specifications for name spaces, linking, database queries and many other ancillary developments are underway at the W3C, using XML. The fundamental HTML is being redone as XHTML (W3C HTMLWorking Group, 1999).

The W3C Math Working Group

It has been rather paradoxical that the mathematical formulae of science have been so difficult to represent on the Web. Tim Berners-Lee, after all, was a scientist at CERN, a massive international center of physics in Geneva, when he came up with what became HTML and the Web.

In May 1997, the W3C chartered a Math Working Group to consider how to facilitate math on the Web. Originally called the HTML-Math WG, it contained representatives of diverse backgrounds. There were people from computer corporations and from publishing, computer algebra people[10] and invited experts from organizations not members[11] of the W3C.

Dave Raggett, the Math WG's W3C staff contact and author of the HTML 3.2 reference specification, was himself an early proponent of adding some math capabilities to HTML. In fact, there was some confusion over math support following HTML 3.2 (Raggett, 1997). Books appeared with sections explaining simple extensions to HTML for math that were little more than suggestions how something might be done. The extensions were not in any Recommendation accepted by the W3C. Recommendations are issued only after a long careful review process, including a period of six weeks during which

[9] Also of importance to math on the Web, especially in education, and for which there is an interest group.

[10] IBM, Hewlett-Packard, Adobe, Elsevier Science, Wolfram Research, Maplesoft, SoftQuad, ...

[11] American Mathematical Society, Geometry Center, Stilo Technologies, (and later Design Science), ...

W3C members may vote for or against acceptance. They normally follow several working drafts, which are made available for public comment. It is a very open process. If there is enough support from the W3C members then a draft is put forward as a Recommendation. The math suggestions were dropped because the matter needed more careful consideration. But this shows the level of interest which was latent in the community.

The Math WG's original far-reaching objectives were listed as follows:

1. is suitable for teaching and scientific communication;
2. is easy to learn and to edit by hand for basic math notation, such as arithmetic, polynomials and rational functions, trigonometric expressions, univariate calculus, sequences and series, and simple matrices;
3. is well suited to template and other math editing techniques;
4. insofar as possible, allows conversion to and from other math formats, both presentational and semantic, such as TEX and computer algebra systems. Output formats may include graphical displays, speech synthesizers, computer algebra systems input, other math layout languages such as TEX, plain text displays (e.g., VT100 emulators), and print media, including braille. It is recognized that conversion to and from other notational systems or media may lose information in the process;
5. allows the passing of information intended for specific renderers;
6. supportes efficient browsing for lengthy expressions;
7. provides for extensibility, for example, through contexts, macros, new rendering schemas or new symbols; some extensions may necessitate the use of new renderers.

The above goals were endorsed by an earlier group meeting in October 1996 at the Boston W3C. Plainly they have not all been accomplished yet but much progress has been made. Point 2, ease of learning and hand editing, at least, cannot be achieved directly with a satisfactorily general and expressive markup language.

Many ideas were considered early on by the WG. TEX is clearly important for communication of mathematics nowadays. Why not just extend HTML with a TEX-like syntax? That turned out not to be simple at all. Finally, after much discussion, the WG decided that it would develop a markup language which accorded with XML. The reason is that it was realized that general acceptance of a new math specification would happen only if it embedded easily into the technology of the internet, which was coming to be dominated by XML and its relatives.

MathML 1.0 is written as an XML application, one of the first at more than a toy level. The Math WG wanted to come up with something that really met the goal of facilitating the use of math on the Web.

Many on the WG had considerable experience with TEX and could consider it as a natural paradigm for a math language for the Web. IBM too, drawing on its extensive experience with Scratchpad and then Axiom, could have itself proposed a language for math. Similarly, Wolfram Research had Mathematica, a very rich language for expressing math in ASCII characters suitable for easy Web transmission.

But there are disadvantages to such foundations for math on the Web. A specification to be generally used for math communication should be publicly developed and not proprietary. Math is almost entirely treated as public property: one cannot, in principle, patent mathematical facts. Agreement is needed from a broad spectrum of interested parties that the language provided is expressive enough for their purposes, for instance, from several symbolic algebra systems. Finally, a specification is more likely to be deployed if those who will use it have been involved in its development. It is also more likely to be realistic if people who will implement it have contributed to its development.

So the Math WG decided to fall in line with the evolving standards of the Web. This was a decision with many implications for our later work and not that easy to take. The primary goal was to create something that would be powerful, usable and adopted. And this aim has continued to drive the WG throughout.

Input considerations

Next the WG set about making an XML application. This had the unfortunate corollary that the markup developed would not directly meet the need for an easy input syntax for math on Web pages, not even for simple math.

After heavy discussion, it was eventually realized that the question of input styles was not one that could be solved initially. There are too many different communities of users of mathematical formulas to satisfy them all. Making a lower-level language, which input mechanisms could write, would encourage development of tools which could be of real help to the many not served by something as

complicated as TEX or LATEX sources. The keyboard input could be accepted by applications tailored to the needs of their user communities — high school, research scientists, computer algebra users, ... In fact, if a symbolic algebra system, or, for instance, Microsoft Word aided by a template input system, can write out MathML markup for equations, then their users do not have to learn a new input environment.

Several tools are already under development. The makers of Mathematica and Maple have pledged their intentions to support MathML in due course, and Design Science has already announced that its template editor, MathType 4.0, will write out MathML (Topping, 1999). IBM's techexplorer (Sutor and Dooley, 1998) already accepts and writes out some MathML in its currently released version.

Conversion of legacy documents

The WG chose a layered approach to facilitating math on the Web. Once an expressive transport protocol is agreed upon then developers can set about making their own compatible tools. In particular, one can make converters of legacy documents into editions using MathML, especially converters of material encoded in TEX. Conversion will never be completely automatic but there are several efforts underway to provide converters. For instance, two talks here will discuss the problems (Gurarie and Rahtz, 1999; Lovell, 1999), and the American Mathematical Society, Society for Industrial and Applied Mathematics and the Geometry Center have funded an on-going project to produce an appropriate tool to deal with the legacy from their TEX publishing systems.

A very simple example

One of the first formulas to try in a new math system is a quadratic equation. Looking closely at that can show quite a lot. So consider a slightly interesting equation[12] and its encoding:

$$x^2 - 79x + 1061 = 0\ . \qquad (1)$$

```
\begin{equation}
    x^2 - 79 x + 1061 = 0\ .
\end{equation}
```

The LATEX is certainly simple. The equation number can be considered to have been provided by some extra-mathematical mechanism. The MathML coding of this display seems verbose by contrast:

MathML presentation coding for equation (1)

```
<math mode="display">
   <mrow>
    <mrow>
       <msup> <mi>x</mi> <mn>2</mn> </msup>
       <mo>-</mo>
       <mrow>
          <mn>79</mn>
          <mo>&InvisibleTimes;</mo>
          <mi>x</mi>
       </mrow>
       <mo>+</mo>
       <mn>1061</mn>
    </mrow>
    <mo>=</mo>
    <mn>0</mn>
   </mrow>
</math>
```

First, note that the punctuational period is not encoded here; this is intentional. The period, and its space away from the equation, should be part of the overall document, not part of the math. MathML is for the math and not for the rest of the document; TEX is a system that can handle both but then tends to mix contexts, as here. TEX is also misused when math mode is employed for special non-mathematical layouts, just as HTML table mechanisms are employed to do math. People should be inventive but there is more to the semantics of the situation than just the displayed form.[13]

A mathematical expression is enclosed within top-level tagging with `<math>`. Almost all the tag names are lowercase; XML is case-sensitive and so a choice was made for MathML. That this piece is to be a "display" as opposed to "in-line" is expressed by the value of an attribute of the `math` element, its `mode`. The values for attributes must be specified, and in quotes, in valid XML.

We see that each leaf of this parse tree, derived from the expression for a quadratic equation, is explicitly labelled as to its element type. All elements are explicitly tagged at start and finish. For math the tagging of conventionally begins with an `m` but that is really only a sop to mnemonics so that the specification is easier to create. `<mo>`, `<mi>` and `<mn>` mean math operator, identifier and number, respectively; identifiers are the sort of thing conventionally set in math italic (variables and so on); exactly what numbers are could be a problem but essentially we

[12] This quadratic has the nice property of delivering prime numbers for integer values of x from 0 to 79; see Mollin (1997).

[13] The period which follows the element tagged with `<math mode="display">` is in the text of the document. The preferred placement of it should be expressed in the accompanying style sheet. How that is done I have not said and using this method depends on style sheets being well implemented. There is the fallback solution of including punctuation as text insertions in math, as is common with TEX.

are thinking of digit strings in some ordinary script. <mrow> provides a grouping construction, which allows an infix notation for operations; the operator here is made explicit with the non-printing character entity ⁢, which is useful for speech rendering and line-breaking with continuation signs. <msup> denotes a special element whose children will be treated differently: the second is a superscript to the first. Of course, the parse tree (fragment), and thus the expression, is represented by the sequence of tokens read in the ordinary manner with condensation of the white space; the pretty-printing is for this exposition.

All this markup provides enough information as input to a screen renderer, or even to a composition system, that it should be able to present the equation correctly. There are many other elements with specialized functions. Let us look at the quadratic formula

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{2}$$

which might be used to find roots of equation (1).

MathML presentation coding for equation (2)

```
<math mode="display">
  <mrow>
   <mi>x</mi> <mo>=</mo>
   <mfrac>
    <mrow>
     <mrow> <mo>-</mo> <mi>b</mi> </mrow>
     <mo>&PlusMinus;</mo>
     <msqrt>
      <mrow>
       <msup> <mi>b</mi> <mn>2</mn> </msup>
       <mo>-</mo>
       <mrow> <mn>4</mn>
        <mo>&InvisibleTimes;</mo> <mi>a</mi>
        <mo>&InvisibleTimes;</mo> <mi>c</mi>
       </mrow>
      </mrow>
     </msqrt>
     </mrow>
     <mrow>
      <mn>2</mn>
      <mo>&InvisibleTimes;</mo> <mi>a</mi>
     </mrow>
   </mfrac>
  </mrow>
</math>
```

Here we see new items: the fraction builder <mfrac>, the square root <msqrt>, a new character entity ± (which is the plus or minus sign), and ⁢ again. To render <mfrac> and <msqrt> a routine has to do some geometry, adjusting lengths of lines to cover subexpressions and providing an appropriate size for the initial part of the square root sign. MathML provides many schemas for placement of symbols in their conventional mathematical relationships; it even extends TEX with, for instance, built-in constructions of pre-super- and pre-subscripts.

The entity ± suggests that a renderer will have to have a whole series of fonts available to it containing representations of the characters — and there will be many of them if the whole of mathematical usage is to be supported.

The Math WG works with the STIX Project set up by the STIPUB group of publishers[14] to identify those characters in use in scientific publishing. It is hoped that arrangements will be made to find places for them in Unicode (Consortium, 1996); the Unicode Technical Committee is considering proposals in this vein. Fonts, preferably publicly and freely available, are then the next priority; STIPUB intends to support their creation, and other people, particularly Taco Hoekwater, are beginning to produce the fonts already.

Presentation and content markup

The last considerations of the previous section bring us back to the ideas mentioned at the start about the significance of signs and the semantics of formulas. So far we have seen *Presentation* markup concerned with capturing the two-dimensional layout of formulas. MathML, because of the strong interests of some of the Math WG members in symbolic computation, attempts to provide a markup language in which more of the semantics of math can be expressed; this is called *Content* markup. Thus the corresponding Content markup for the two expressions above is different (see the next column).

We see that equation (1) as a whole has been identified as an equality relationship by the surrounding element <reln> and its first empty child element <eq />. It is an equality between the result of a function application, shown by <apply> and a number, identified as such by <cn>. The Content number element <cn> has to be distinguished from the <mn> we have already met; assumptions about it may be made which may be useful to mathematics processing systems. The type of function being applied is shown by <plus />; it applies to a sequence of elements, two <apply>s and a number. The functions

[14] STIPUB stands for "Scientific and Technical Information Publishers," a committee whose members include representatives from several learned societies and publishers. They meet from time to time to consider matters of mutual interest. STIX stands for the working group they set up to consider "Scientific and Technical Information eXchange" insofar as it concerns the characters that should be in Unicode and a set of fonts adequate to display them; see www.ams.org/stix/.

applied are `<power/>` and `<minus/>`, which are, respectively, binary and unary functions.

MathML content coding for equation (1)

```
<math mode="display">
<reln>
   <eq />
   <apply>
     <plus />
     <apply>
       <power />
       <ci>x</ci>
       <cn>2</cn>
     </apply>
     <apply>
       <minus />
       <apply>
         <times />
         <cn>79</cn>
         <ci>x</ci>
       </apply>
       <ci>x</ci>
     </apply>
     <cn>1061</cn>
   </apply>
   <cn>0</cn>
</reln>
</math>
```

This alternative way of marking up the expression is intended to assist computer algebra systems and search engines looking for mathematical expressions given the semantics. Content markup does not necessarily bring with it an immediately clear visually rendered form; at the very least, transformation rules need to be supplied to produce an appropriate Presentation markup. MathML has made provision for the addition of assertions about preferred presentation of Content markup and about the content semantics of Presentation markup. They can be used together but it is naturally not that easy to combine the two.

For the quadratic formula we have something similar: a relationship of equality between results of a cascade of function applications (see the next column). The entity `&PlusMinus;` occurs again but this time made into a function label by being the content of an `<fn>` element. Otherwise, the only new items are function labels: `<divide />`, `<times />`.

In an attempt to allow the capture of much of the semantics of elementary math, some 75 or so Content elements are provided in MathML. This number may change during the revision of MathML to version 2.0, which the present Math WG is undertaking. There remains discussion as to how best to capture content semantics and how much to include. In fact, the next version has to include extension mechanisms, for presentation and content markup, and for symbols. so the users may extend the language to cover what was not thought of. The watchword thus far has been what is known in the US as K–14 math education.[15]

MathML content coding for equation (2)

```
<math mode="display">
<reln>
   <eq />
   <ci>x</ci>
   <apply>
     <divide />
     <apply>
       <fn><mo>&PlusMinus;</mo></fn>
       <apply>
         <minus />
         <ci>b</ci>
       </apply>
       <apply>
         <root />
         <apply>
           <minus />
           <apply>
             <power />
             <ci>b</ci>
             <cn>2</cn>
           </apply>
           <apply>
             <times />
             <cn>4</cn>
             <ci>a</ci>
             <ci>c</ci>
           </apply>
         </apply>
         <cn>2</cn>
       </apply>
     </apply>
     <apply>
       <times />
       <cn>2</cn>
       <ci>a</ci>
     </apply>
   </apply>
</reln>
</math>
```

Conclusion

Looking at these simple examples only scratches the surface of a markup language with about 150 primitive elements and ten times that many identified primitive character entities. The devil is in the

[15] K–14 means two college years beyond Kindergarten through grade 12 in the US educational system. This is not an exact range specified by any educational norms. The final level may correspond in Europe to matters covered in lycée, Gymnasium or college, say.

details, as usual. Examples of formulas can be discussed much faster when speaking in front of overheads than on the printed page.

The WG is issuing a corrected version, MathML 1.01, in July 1999 and will provide a major revision and extension, MathML 2.0, by February 2000. Better integration with all the new standards from W3C will be part of version 2.0 and extensibility issues will be further addressed.

In the meantime, the promising new development is that the largest browser companies are beginning to implement MathML rendering in an essentially native way. Up to the present the best display of MathML with a browser has been with the Java plug-in WebEQ (which also has an associated equation editor), or with the W3C's testbed browser Amaya. The Math WG is also working on providing a test suite to verify compliance with the specification issued and to help those who build tools using MathML. There is a great deal of activity of MathML development in process.

Acknowledgements

The activity of a W3C Working Group is very much a collaborative one. Aside from the numerous people who have expressed their views on how math should be on the Web, I would like very much to thank all the members of the W3C Math WG — Stephen Buswell, David Carlisle, Stéphane Dalmas, Stan Devitt, Ben Hinkle, Angel Díaz, Sam Dooley, Stephen Hunt, Roger Hunter, Doug Lovell, Robert Miner, Barry MacKichan, Ivor Philips, Nico Poppelier, T.V. Raman, Dave Raggett, Murray Sargent III, Neil Soiffer, Paul Topping, Stephen Watt, the current members, and the past members, Stephen Glim, Brenda Hunt, Arnaud Le Hors, Bruce Smith, Robert Sutor, Ron Whitney, Lauren Wood, Ka-Ping Yee, Ralph Youngen, for the pleasure and stimulation of working with them. Together they have achieved something very worthwhile.

In addition, we are all very grateful to Barbara Beeton for her stalwart efforts in trying to get the characters of math into Unicode, and thus onto the Web. And I am grateful to the American Mathematical Society for supporting my efforts here, and to the IHES for superlative conditions during a long visit there.

References

Apparao, V., S. Byrne, M. Champion, S. Isaacs, I. Jacobs, A. L. Hors, G. Nicol, J. Robie, R. Sutor, C. Wilson, and L. Wood, eds. "Document Object Model (DOM) Level 1 Specification". Technical report, World Wide Web Consortium, `http://www.w3c.org/TR/REC-DOM-Level-1-19981001`, 1998. The latest version of DOM1 is available at `http://www.w3c.org/TR/REC-DOM-Level-1`.

Bos, B., H. W. Lie, C. Lilley, and I. Jacobs, eds. "CSS, level 2 Recommendation". Technical report, World Wide Web Consortium, `http://www.w3c.org/TR/REC-CSS2-19990512`, 1998. The latest version of CSS2 is available at `http://www.w3c.org/TR/REC-CSS2`.

Bos, B. and H. W. Lie, eds. "CSS, level 1 Recommendation". Technical report, World Wide Web Consortium, `http://www.w3c.org/TR/REC-CSS1-19990111`, 1999. The latest version of CSS1 is available at `http://www.w3c.org/TR/REC-CSS1`.

Bray, T., J. Paoli, and C.M. Sperberg-McQueen, eds. "Extensible Markup Language (XML) 1.0.". 1998. The latest version of XML is available at `http://www.w3c.org/TR/REC-xml`.

Cajori, Florian. *A History of Mathematical Notation*. Open Court Publishing, La Salle, Illinois, 1928/29. 2 vols. Reprinted (Cajori, 1993).

Cajori, Florian. *A History of Mathematical Notation*. Dover, New York, 1993. 2 vols. printed together.

Deach, Stephen, ed. "Extensible Stylesheet Language (XSL) Specification". Technical report, World Wide Web Consortium, `http://www.w3c.org/TR/1999/WD-xsl-19990421/`, 1999. The latest version of the XSL Working Draft is available at `http://www.w3c.org/TR/WD-xsl`.

DSSSL. *Document Style Semantics and Specification Language (DSSSL)*. ISO/IEC 10179:1996(E), 1996. Available at `ftp://ftp.ornl.gov/pub/sgml/WG8/DSSSL/`.

Fielding, R., J. Gettys, J. Mogul, H. F. Nielsen, T. Berners-Lee, et al.. "HTTP Version 1.1". Technical report, Internet Engineering Task Force (IETF), 1998.

Godement, Roger. *Analyse Mathématique I*. Springer, Berlin, Heidelberg and New York etc., 1998.

Gurarie, Eitan and S. Rahtz. "LaTeX to XML/MathML (Workshop)". *TUGboat* **20**(3), 1999. (See elsewhere in these Proceedings.).

Ion, Patrick D.F. and R. L. Miner, eds. "Mathematical Markup Language (MathML) 1.0 Specification". Technical report, World Wide Web Consortium, `http://www.w3c.org/TR/REC-MathML-19970430`, 1997. The latest version of MathML is available at `http://www.w3c.org/TR/REC-MathML`.

ISO 8879:1986. *Information Processing — Text and Office Systems — Standard Generalized Markup Language (SGML)*. International Standards Organization, ftp://ftp.ornl.gov/pub/sgml/WG8/DSSSL/, 1986. Consult http://www.iso.ch/cate/d16387.html for information about the standard.

Lassila, O. and R. Swick, eds. "Resource Description Framework (RDF) Model and Syntax Specification". Technical report, World Wide Web Consortium, http://www.w3c.org/TR/REC-rdf-syntax-19990222, 1999. The latest version of RDF is available at http://www.w3c.org/TR/REC-rdf-syntax.

Lovell, Doug. "TEXML brings TEX to Web Future". *TUGboat* **20**(3), 1999. (See elsewhere in these Proceedings.).

Mollin, R.A. "Prime-producing quadratics". *Amer. Math. Monthly* **104**(6), 529–544, 1997. MR 98h:11113.

Peirce, C.S. "Description of a Notation for the Logic of Relatives". *Memoirs Acad. of Arts and Sciences* (Cambridge and Boston; N.S.) **IX**, 317–378, 1867.

PICS. "Platform for Internet Content Selection (PICS)". http://www.w3c.org, 1997. For more information see http://www.w3.org/PICS/.

Raggett, D., A. Le Hors and I. Jacobs, eds. "HyperText Markup Language (HTML) 4.0 Specification". Technical report, World Wide Web Consortium, http://www.w3c.org/TR/REC-html40-19980424, 1998. The latest version of HTML 4.0 is available at http://www.w3c.org/TR/REC-html40.

Raggett, D., ed. "HTML 3.2 Reference Specification". Technical report, World Wide Web Consortium, http://www.w3c.org/TR/REC-html32, 1997. The latest version, HTML 4.0, is available at http://www.w3c.org/TR/REC-html40.

Rudio, Ferdinand, 1856-1929., editor. *Verhandlungen des ersten internationalen Mathematiker-Kongresses in Zürich vom 9. bis 11. August 1897 (Congrès International des Mathématiciens; International Congress of Mathematicians)*, Leipzig. B.G. Teubner, 1898.

Schröder, (Friedrich Wilhelm Karl) Ernst, 1841-1902. "Über Pasigraphie, ihren gegenwärtigen Zustand und die pasigraphische Bewegung in Italien". In *Verhandlungen des ersten internationalen Mathematiker-Kongresses*, pages 147–162. 1898.

Sutor, Robert S. and S. S. Dooley. "TEX and LATEX on the Web via IBM techexplorer". *TUGboat* **19**(2), 157–161, 1998.

Topping, Paul R. "Using MathType to Create TEX and MathML Equations". *TUGboat* **20**(3), 1999. (See elsewhere in these Proceedings.).

Unicode. *The Unicode Standard: Version 2.0*. The Unicode Consortium, 1996. The specification also takes into consideration the corrigenda at http://www.unicode.org/unicode/uni2errata/bidi.htm. For more information, consult the Unicode Consortium's home page at http://www.unicode.org/.

van Herwijnen, Eric. *Practical SGML*. Kluwer Academic Publishers Group, Norwell and Dordrecht, 1994.

W3C HTML Working Group. "XHTML 1.0: The Extensible HyperText Markup Language; A Reformulation of HTML 4.0 in XML 1.0". Technical report, World Wide Web Consortium, http://www.w3.org/TR/1999/xhtml1-19990505, 1999. This is a Working Draft; the latest version of XHTML is available at http://www.w3c.org/TR/xhtml1.

W3C Style. "Home page of the W3C Style Activity". http://www.w3.org/Style/Activity , 1999. This includes XSL and CSS work.

TEXML: Typesetting XML with TEX

Douglas Lovell
IBM Research
P.O. Box 704
Yorktown Heights, NY 10598
dcl@us.ibm.com

Abstract

XML, eXtensible Markup Language, is a simplified subset of SGML, which is fast becoming a standard for content management on the internet.

TEXML is an XML vocabulary for TEX. A processor written in the Java programming language translates TEXML-conforming XML into TEX. The processor provides a document formatting solution for XML that leverages the rich knowledge and capability built over many years in TEX.

This describes the TEXML document format and the processor, TEXMLattè, that produces TEX source from TEXML markup.

XML: The future of the Web

The World Wide Web is moving toward a future in which XML, not HTML, is the primary medium for storing and delivering documents and data.

HTML is presentation markup for browsers. Together with Cascading Style Sheets (CSS) it specifies the type sizes and fonts, and layout of a document.

XML is a standard for defining and sharing data on the Web, including Web content. Users of XML may define vocabularies — sets of tags or element names, such as "title," "section," "citation" and attributes such as "id" — to identify elements and structures in a document. With XML, organizations can develop markup which captures the structural and semantic properties of their documents and data. Instead of writing, `<H1>Typesetting XML</H1>` we can write, `<title>Typesetting XML </title>`.

Trade organizations, companies, and scientific or educational institutions may define and share XML vocabularies to freely exchange data with specific meaning. MathML (W3C, 1998), the vocabulary for writing and exchanging mathematical formulas and expressions, is one example.

In the commercial world, XML is a key technology for the widespread implementation of rapid, accurate, meaningful, automatic transactions and data exchange on the internet.

What is XSL?

XSL, eXtensible Stylesheet Language, is a W3C draft recommendation (XSLWorking Group W3C, 1998) which began life as an XML vocabulary for typesetting. XSL is influenced by the DSSSL standard (ISO/IEC, 1996).

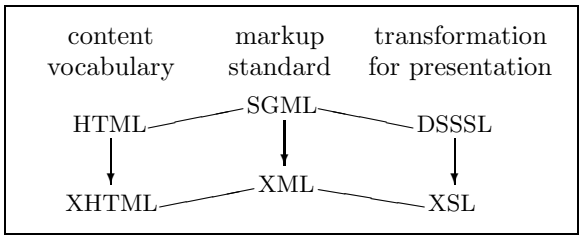


Figure 1: XML heritage

DSSSL is an ISO standard for typesetting SGML. The XSL effort began as a means for transforming XML markup into standard presentation markup, just as DSSSL did for SGML. A goal for XSL was to write a shorter, more perspicuous standard in the spirit of XML. Some of the key people who worked on DSSSL are key people working on XSL.

XSL does two things for XML:

1. It provides a language for specifying transformations from one XML vocabulary into another. The XSL specification calls this "Tree Construction."
2. It defines a set of XML elements, called "formatting objects", for encoding a typeset document specification in XML.

Figure 1 diagrams the heritage of XML and XSL from SGML and DSSSL. HTML is an SGML vocabulary which is migrating to an XML vocabulary, XHTML. XML is a content markup standard with ancestry in SGML. XSL is a formatting and transformation standard for XML with ancestry in DSSSL.

```
<?xml version='1.0'?>
<artist>
 <id>Weston</id>
 <fullName>Edward Weston</fullName>
 <lastName>Weston</lastName>
 <firstName>Edward</firstName>
 <birthYear>1886</birthYear>
 <deathYear>1958</deathYear>
 <portrait>Weston.jpg</portrait>
 <bio>Weston was a photographer who pioneered the modern use of photography
  as an art form in the United States.</bio>
 <longQuote1>One does not think during creative work, any more than one thinks
  when driving a car.  But one has a background of years - learning,
  unlearning, success, failure, dreaming, thinking, experience, all
  this - then the moment of creation, the focusing of all into the
  moment.  So I can make 'without thought,' fifteen carefully
  considered negatives, one every fifteen minutes, given material
  with as many possibilities.  But there is all the eyes have seen in
  this life to influence me.</longQuote1>
 <shortQuote1>My own eyes are no more than scouts on a preliminary search,
  for the camera's eye may entirely change my idea.</shortQuote1>
 <shortQuote2>Ultimately success or failure in photographing people depends
  on the photographer's ability to understand his fellow man.</shortQuote2>
 <soundQuote>Weston.wav</soundQuote>
</artist>
```

Figure 2: XML markup describing an artist

Figure 2 shows a sample of some XML markup describing an artist, taken from an application for an art museum. For more about XML see IBM/XML (1999) and W3C (1999).

What is TEXML?

TEXML is an XML vocabulary for representing TEX source. It is a medium for transforming any XML data into a document which can be typeset with the TEX program. TEXML represents the TEX commands, control symbols, and `\special`s as XML elements. Figure 3 shows the markup of Figure 2 represented in TEXML.

The potential for TEX and XML

XML makes TEX a potential universal typesetting back-end for markup in a way it never achieved for SGML. There is an opportunity for TEX to marry itself to XML, the future of the WWW, and become a predominant technology for typesetting.

TEX has many advantages. It has an established, stable implementation and user base. It has a rich body of knowledge and experience captured in its macro packages and styles. It can typeset just about anything.

The missing piece is a small implementation of TEX in the Java programming language which can be plugged into a browser or executed on a server; however, much of LATEX can be displayed by IBM techexplorer (Sutor and Díaz, 1998).

The enabling technologies now available are TEXML and XSL, with platform-specific implementations of TEX or with the IBM techexplorer plug-in for an HTML browser.

XSL Tree Construction. The transformation part of XSL has been moving rapidly toward completion as a W3C recommendation. There are many implementations of the XSL transformation rules. In the process, people have found uses for XSL transforms far beyond producing typeset documents.

There are numerous web servers busily transforming XML into HTML using XSL style sheets. Microsoft is supplying the transform as a dynamic link library in their operating system. The Microsoft Internet Explorer Web browser (v.5) can accept XML data, apply an associated XSL transform, and display the result. Organizations use XSL style sheets to transform electronic data from another organization into a form suitable for internal use.

```
<TeXML>
 <cmd name="documentclass">
  <parm>article</parm>
 </cmd>
 <cmd name="title">
  <parm>Edward Weston</parm>
 </cmd>
 <env name="document">
  <cmd name="maketitle"/>
  <cmd name="section*">
   <parm>Some biographical information</parm>
  </cmd>
  Edward Weston
  was born in
  1886.
  Weston
  lived until
  1958.
  <cmd name="par"/>
  Weston was a photographer who pioneered the modern use of photography
  as an art form in the United States.
  <cmd name="section*">
   <parm>Some short quotes from Weston</parm>
  </cmd>
  <env name="enumerate">
   <cmd name="item"/>
   ‘‘My own eyes are no more than scouts on a preliminary search,
   for the camera’s eye may entirely change my idea.’’
   <cmd name="item"/>
   ‘‘Ultimately success or failure in photographing people depends
   on the photographer’s ability to understand his fellow man.’’
  </env>
  <cmd name="section*">
  <parm>A longer quote</parm>
  </cmd>
  <env name="quote">
   ‘‘One does not think during creative work, any more than one thinks
  when driving a car.  But one has a background of years - learning,
  unlearning, success, failure, dreaming, thinking, experience, all
  this - then the moment of creation, the focusing of all into the
  moment.  So I can make ’without thought,’ fifteen carefully
  considered negatives, one every fifteen minutes, given material
  with as many possibilities.  But there is all the eyes have seen in
  this life to influence me.’’
  </env>
 </env>
</TeXML>
```

Figure 3: TEXML markup derived from the XML of Figure 2

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">

<xsl:template match="artist">
 <TeXML>
  <cmd name="documentclass"><parm>article</parm></cmd>
  <cmd name="title">
   <parm><xsl:apply-templates select="fullName"/></parm>
  </cmd>
  <env name="document">
   <cmd name="maketitle"/>
   <cmd name="section*">
    <parm>Some biographical information</parm>
   </cmd>
   <xsl:apply-templates select="birthYear"/>
   <xsl:apply-templates select="deathYear"/>
   <cmd name="par"/>
   <xsl:apply-templates select="bio"/>
   <cmd name="section*">
    <parm>Some short quotes from
     <xsl:apply-templates select="lastName"/>
    </parm>
   </cmd>
   <env name="enumerate">
    <cmd name="item"/>
    ``<xsl:apply-templates select="shortQuote1"/>''
    <cmd name="item"/>
    ``<xsl:apply-templates select="shortQuote2"/>''
   </env>
   <cmd name="section*">
    <parm>A longer quote</parm>
   </cmd>
   <env name="quote">
   ``<xsl:apply-templates select="longQuote1"/>''
   </env>
  </env>
 </TeXML>
</xsl:template>

<xsl:template match="birthYear">
 <xsl:apply-templates select="//fullName"/>
 was born in
 <xsl:apply-templates/>.
</xsl:template>

<xsl:template match="deathYear">
 <xsl:apply-templates select="//lastName"/>
 lived until
 <xsl:apply-templates/>.
</xsl:template>

</xsl:stylesheet>
```

Figure 4: XSL markup to transform XML for an artist into TEXML

```
<?xml version='1.0'?>
<!--* DTD for translation to TeX *-->

<!ENTITY % content "#PCDATA|cmd|env|ctrl|spec">

<!ELEMENT TeXML (%content;)*>

<!ELEMENT cmd (opt|parm)*>
<!ATTLIST cmd name CDATA #REQUIRED>

<!ELEMENT opt (#PCDATA|cmd|ctrl|spec)*>

<!ELEMENT parm (#PCDATA|cmd|ctrl|spec)*>

<!ELEMENT env (%content;)*>
<!ATTLIST env name CDATA #REQUIRED>
<!ATTLIST env begin CDATA #IMPLIED>
<!ATTLIST env end CDATA #IMPLIED>

<!ELEMENT ctrl EMPTY>
<!ATTLIST ctrl ch CDATA #REQUIRED>

<!ELEMENT group (%content;)*>

<!ELEMENT spec EMPTY>
<!ATTLIST spec cat (esc|bg|eg|mshift|align|parm|sup|sub|comment|tilde) #REQUIRED>
```

Figure 5: The DTD for TEXML

XSL has thus become an important tool for XML transformation. It is on the verge of becoming a *de facto* standard for XML transformations.

Figure 4 shows an XSL stylesheet that will transform the artist XML example in Figure 2 into the TEXML example in Figure 3.

XSL Formatting Objects. The formatting objects (FO) part of XSL is progressing more slowly. The specification of FO elements and the structure of those elements is far behind the transformation part as of this writing (March, 1999). This means that, while there is a "standard" way to transform XML into HTML, there is presently no standard way to produce typeset output from XML. TEXML was developed partly out of impatience to fill this gap left by the lagging FO effort. It marries the transformation function of XSL with the typesetting function of TEX. It provides a means for typesetting XML documents.

How to use TEXML

TEXML consists of two parts. The first part is the document type declaration (DTD), which defines XML that is valid TEXML. The second part is the translator program TEXMLattè, written in the Java programming language. TEXMLattè reads a valid TEXML file and writes a proper TEX file. The TEXML DTD appears as Figure 5.

The basic template for a TEXML document is:

```
<?xml version="1.0"?>
<TeXML>
  ... your content here ...
</TeXML>
```

The following sections describe how the DTD and TEXMLattè interact, and how to code TEX in TEXML.

Encoding commands. The TEXML <cmd> element encodes TEX commands.

1. To write a command with no parameters, such as \par, write <cmd name="par"/>.
2. To add parameters to a command, add <parm> children[1] to the <cmd> element. TEXMLattè places <parm> children within TEX groups, that is, curly braces.

[1] XML elements embedded within an enclosing element are often referred to as that element's "children," e.g. <parent><child/></parent>.

TEXML	TEX
%	`\%{}`
&amp;	`\&{}`
{	`\{`
}	`\}`
|	`$\|$`
\	`$\backslash$`
$	`\${}`
#	`\#{}`
_	`\_{}`
^	`\char'\^{}`
~	`\char'\~{}`
&lt;	`$<$`
&gt;	`$>$`

Figure 6: Characters escaped by TEXMLattè

3. To add options to a command, add <opt> children to the <cmd> element. TEXMLattè places <opt> children within square braces, as LATEX style options.

As an example, the TEX code

```
\documentclass[12pt]{letter}
```

will look like this in XML:

```
<cmd name="documentclass">
  <opt>12pt</opt>
  <parm>letter</parm>
</cmd>
```

The TEXML DTD allows free-form text, commands, control symbols, and \specials as children of <parm> and <opt> elements. It does not allow <parm> or <opt> elements to nest, except as children of a nested <cmd>. It does not allow environments within a <parm> or an <opt>.

Encoding control symbols. The <ctrl> element encodes a control symbol, such as <ctrl ch=" "/> for a control space. Use the <cmd> element to encode control words.

Encoding specials. TEXMLattè "escapes" any and all TEX \specials which occur in the XML source. This means that backslashes, percent signs, dollar signs, and the like are properly escaped by the time they get to TEX. The writer of XML will not intend those characters to be special. Figure 6 gives a full list of the characters escaped by TEXMLattè, along with their replacement text.

Use the <spec> element to encode any TEX \specials when you really want them to occur as \specials in the TEX file output by TEXMLattè.

The <spec> element is always empty; that is, it never has anything within it. Encode the category of the special from Figure 7 in the required "ch" attribute (e.g. <spec ch="bg"/>). The translator will output the character indicated in the table. It is possible to foil this by changing the category of the character output by the translator, so beware.

description	ch attribute	output
escape character	esc	\
begin group	bg	{
end group	eg	}
math shift	mshift	$
alignment tab	align	&
parameter	parm	#
superscript	sup	^
subscript	sub	_
tilde	tilde	~
comment	comment	%

Figure 7: <spec> "ch" attribute values

End-of-line characters (TEX category 5) are treated as space by XML. TEXMLattè substitutes a space (TEX category 10) character for the end-of-line character. There is no way to encode the end-of-line character in TEXML. Paragraph breaks must be coded with <cmd name="par"/>.

There is no need to encode the ignored character (TEX category 9). If it is to be ignored, there is no reason to put it in. The same is true of the invalid character (TEX category 15).

Encoding environments. The element <env> is a convenience for expressing LATEX environments. To have TEXMLattè output:

```
\begin{document}
...
\end{document}
```

we write in TEXML:

```
<env name="document">
  ...
</env>
```

The <env> element is not strictly required. It is supplied because it correctly captures in XML the spirit of an environment: it opens a context which is later closed. It is also much more convenient to use than the alternative method, using the <cmd> element:

```
<cmd name="begin">
  <parm>document</parm>
</cmd>
...
<cmd name="end">
  <parm>document</parm>
</cmd>
```

TEXMLattè supplies the default values, "begin" for the begin attribute, and "end" for the end attribute of an <env>. If you have an environment which uses a different convention for the begin and end commands you will specify the "begin" and "end" attributes. For example, the following produces an environment that begins with \s{t} and ends with \e{t}.

```
<env name="t" begin="s" end="e">
  ET phone home!
</env>
```

Any TEXML content may appear within an environment.

Encoding groups. The <group> element is a convenience for encoding groups.

TEXMLattè will supply an open brace at the beginning, and a close brace at the end of the group. The TEX scrap, {\it italics} may appear as <group><cmd name="it"/>italics</group>.

This is much easier to write than

```
<spec cat="bg"/>
  <cmd name="it"/>
  italics
<spec cat="eg"/>
```

There is some technical advantage as well as convenience. The <group> element allows the XML parser to catch any missing close brace by the absence of the close group (</group>). XML cannot detect a missing <spec cat="eg"/>.

Any TEXML content may appear within a group.

Conclusion

XML is taking the world by storm. IBM is committed to making XML a viable and widely adopted standard for engaging in electronic commerce and publishing on the internet.

IBM's TEXML provides an immediate solution for typesetting any XML content. It positions TEX as a potential typesetting back-end for internet publishing and electronic commerce applications.

Figure 8 displays the TEX resulting from processing the XML of Figure 3 with TEXMLattè.

A How to get TEXML

TEXML is available for download from the IBM AlphaWorks website:

http://www.alphaworks.ibm.com

Look for TeXML. You will find full source, instructions, and examples at the site.

You will need the following in addition to the files provided there:

An XSL implementation. We have used the Lotus implementation, which you may download from the IBM AlphaWorks web site. Look for LotusXSL. You will find references to more implementations at the W3C XSL website:

http://www.w3.org/Style/XSL/

A JAVA run-time implementation. You will find these at the JavaSoft website:

http://www.javasoft.com

Look for Java runtime under products.

An XML parser. TEXMLattè uses the XML4J parser available from the IBM AlphaWorks web site. Look for xml4j.

A LATEX implementation. You will find references to these at the TEX Users Group website:

http://www.tug.org/

References

IBM/XML. "IBM Answers your XML Questions". 1999. See http://www.ibm.com/xml/.

ISO/IEC. "Information technology — Processing languages — Document Style Semantics and Specification Language (DSSSL)". International Standard ISO/IEC 10179:1996(E), International Standards Organization, 1996. See http://www.oasis-open.org/cover/dsssl.html.

W3C. "Extensible Markup Language (XML) Activity". 1999. See http://www.w3.org/XML/Activity.html.

W3C, MathML Working Group. "Mathematical Markup Language (MathML) 1.0". W3C Recommendation REC-MathML-19980407, W3C, 1998. See http://www.w3.org/TR/REC-MathML-1980407.html.

XSL Working Group W3C. "Extensible Stylesheet Language (XSL) Version 1.0". W3C Working Draft WD-xsl-19981216, W3C, 1998. See http://www.w3.org/TR/1998/WD-xsl-19981216.html.

Sutor, Robert S. and A. L. Díaz. "IBM techexplorer: Scientific Publishing for the Internet". In *Proceedings of the EuroTEX'98 Conference, St. Malo, France*, volume 28/29, pages 295–308. 1998. Also avail. at: http://www.gutenberg.eu.org/pub/GUTenberg/publications/publis.html.

```
\documentclass{article}
\title{Edward Weston}
\begin{document}
 \maketitle
 \section*{Some biographical information}
 Edward Weston
 was born in
 1886.
 Weston
 lived until
 1958.
 \par
 Weston was a photographer who pioneered the modern use of photography
 as an art form in the United States.
 \section*{Some short quotes from Weston}
 \begin{enumerate}
  \item
  ``My own eyes are no more than scouts on a preliminary search,
  for the camera's eye may entirely change my idea.''
  \item
  ``Ultimately success or failure in photographing people depends
  on the photographer's ability to understand his fellow man.''
 \end{enumerate}
 \section*{A longer quote}
 \begin{quote}
  ``One does not think during creative work, any more than one thinks
 when driving a car.  But one has a background of years - learning,
 unlearning, success, failure, dreaming, thinking, experience, all
 this - then the moment of creation, the focusing of all into the
 moment.  So I can make 'without thought,' fifteen carefully
 considered negatives, one every fifteen minutes, given material
 with as many possibilities.  But there is all the eyes have seen in
 this life to influence me.''
 \end{quote}
\end{document}
```

Figure 8: The TEX produced by TEXMLattè using the XML of Figure 3

Using MathType to Create TEX and MathML Equations

Paul Topping
Design Science, Inc.
4028 Broadway
Long Beach, CA 90803
USA
pault@mathtype.com
www.mathtype.com

Abstract

MathType 4.0 is the latest release of Design Science's interactive mathematical equation editing software package, the full-featured version of the Equation Editor applet that comes with Microsoft Word. Its completely re-architected translation system should be of particular interest to the TEX and MathML communities. MathType can be used as an aid to learning TEX, as a simpler interface for entering equations into a TEX authoring system, or as part of a document conversion scheme for journal and book publishers.

After the introduction, a simple translation example is given to show how its Translator Definition Language (TDL) is used to convert a MathType equation to TEX. This is followed by an introduction to MathML and MathType's MathML translators are discussed. Finally, some of the possibilities for creating translators for special purposes is mentioned, along with discussion on how MathType's translation facilities can be used as component of a more comprehensive document conversion process.

Introduction

MathType is an interactive tool for authoring mathematical material. It runs on Microsoft Windows and Apple Macintosh systems (a Linux implementation is under consideration). Readers may also be familiar with MathType's junior version, Equation Editor, as it is supplied as part of many personal computer software products, such as Microsoft Word and Corel WordPerfect.

Unlike TEX, MathType does not process entire documents. Rather, it is used in conjunction with other products, such as word processors, page layout programs, presentation programs, web/HTML editors, spreadsheets, graphing software, and virtually any other kind of application that allows insertion of a graphical object into its documents. MathType equations can even be inserted into database fields with most modern database systems!

MathType has a simple but powerful direct-manipulation interface for creating standard mathematical notation. Instead of entering a computer language, such as TEX, the MathType user combines simple typing with the insertion of "templates". For example, inserting a fraction template results in a fraction bar with empty slots above and below for the numerator and denominator. The contents of each slot are filled in by the user by more typing and inserting of templates. The displayed equation is reformatted as the user types and spacing is added automatically (although spacing may be explicitly overridden). Some find this interface to be simpler than direct TEX input as there are no keywords to

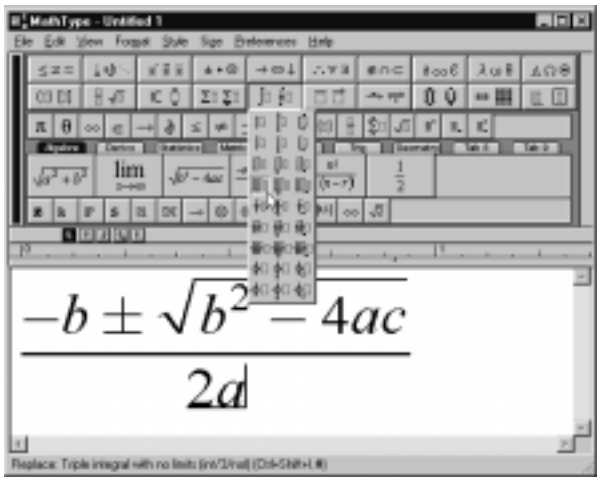


Figure 1: The MathType Window

remember and, most importantly, no possibility of syntax errors.

One common complaint heard from TEX users upon first seeing MathType's user interface is that one must use the mouse for everything. Whereas mouse support is an important part of MathType's user interface (as with virtually all Windows and Mac applications), MathType 4.0 has keyboard methods for performing all of its commands. Also, users may assign keystrokes to commands in any kind of mnemonic scheme they care to invent. The ability to assign a keystroke to an arbitrary math expression is analogous to TEX's macro facility.

Although a thorough examination of MathType is beyond the scope of this paper, here are some of its most important features:

- Any national language characters that the host operating system allows may be inserted into math, including Asian characters.
- MathType's internal representation of characters is Unicode,[1] extended via its Private Use area to cover more of the characters that appear in mathematical notation. We call this MTCode. A user-extendable database of math font-to-MTCode mappings is used to relate characters entered to knowledge used in line formatting, as well as translation.
- A basic set of mathematical fonts (Roman, Greek, italics, Fraktur, blackboard bold, and many mathematical symbols) is included. MathType can also make use of any PostScript Type 1 or TrueType font available via the operating system.
- MathType also includes a translation system for converting mathematics entered in its editing window to virtually any text-based language. This translation system is the chief subject of this paper. In particular, it includes translators for several flavors of TEX and MathML.

MathType's translation facilities

MathType has had a TEX translator for many years that allows the user to copy all or part of an expression onto the clipboard in the TEX language, ready to be pasted into a document. However, until version 4.0, it had two important limitations: it could only generate plain TEX and the user had no control over the TEX fragments generated for particular symbols and templates. MathType 4.0 features a complete re-design of the translator mechanism. The translation of a MathType expression is controlled by a translator definition file, a text file containing a simple translation rule language that allows a fragment of the target language to be associated with each of MathType's many symbols and templates. Although the chief motivation for its development was TEX translation, it can also be used to convert MathType equations to other languages, such as MathML and those specified by the math parts of various SGML-based document languages.

MathType is supplied with translator definition files for plain TEX, $\mathcal{AMS}$-TEX, LATEX, $\mathcal{AMS}$-LATEX, and four MathML variations. These can be customized for specific applications or translators for other mathematical languages can be written by starting from scratch. Also, commands are available in Microsoft Word that will allow a Word document containing MathType (or Equation Editor) equations to be converted to TEX or any other language supported by a translator. MathType's translation facilities can be used as an aid to learning TEX, as a simpler interface for entering equations into a TEX authoring system, or as part of a document conversion scheme for journal and book publishers.

The MTCode character encoding

Although the designers of Unicode have attempted to include many mathematical characters, their attempt falls somewhat short. In fairness to them, incorporating all the characters of the many natural languages in use in the world must have been an overwhelming task.

There is an attempt by some in the mathematical community to get the Unicode Consortium to add the missing mathematical characters to a future version of Unicode. If and when they are successful, we will probably adopt it to replace MTCode.

MathType uses each character's MTCode value as a key into a database of character information that, for each character, includes a human-readable description, an indicator of its role in mathematical notation (e.g. variable, binary operator), and information used in the process of choosing an appropriate font to render it on screen and printer. Most importantly, a character's MTCode value is an index into tables of translation strings in MathType's translation system.

We will use the terms MTCode and Unicode interchangeably in the remainder of this paper.

[1] Unicode is a standard for encoding characters. See `www.unicode.org` for information.

The translation system from a user's perspective

The basic scenario for using MathType to aid in the creation of a TEX document is to run it simultaneously with your favorite TEX editor. The process is this:

- whenever an equation is needed, the MathType window is brought to the front;
- the equation is created in the MathType window;
- the equation is selected and copied to the clipboard, a process which invokes the previously selected translator;
- the TEX editor is brought to the front;
- the TEX code for the equation is pasted into the document.

Editing to correct mistakes is performed by reversing this process, pasting the TEX code (along with a comment containing a compressed form of MathType's internal representation) back into a MathType window, and then repeating the above process once the corrections have been made.

At the beginning of such a document creation and editing session, the user must select one of MathType's translators. This is done via the Translators dialog, which presents a list of all the translators present on the user's system in the MathType translators directory (Fig. 2).

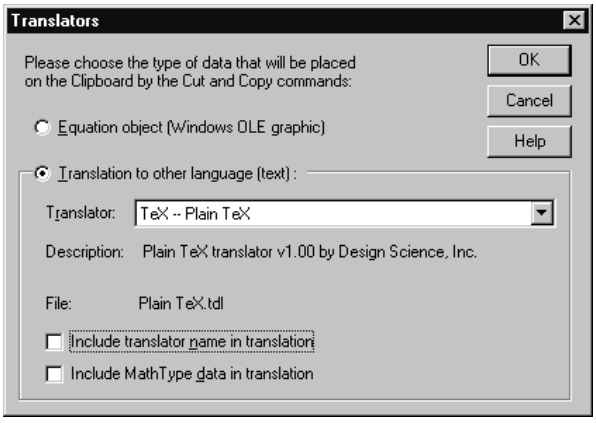


Figure 2: The Translators dialog

Anatomy of a translator

Each translator is defined by a text file written in a simple language called TDL (Translator Definition Language). A translator has a simple structure:

- The first line defines the short name for the translator which appears in the list presented to the user in the Translators dialog (see Fig. 2) and a longer description which appears in the dialog once the translator is chosen from the list. The description might include the author's name and affiliation as well as the version number of the translator.
- a set of matching rules of the form
  $\langle$*thing to match*$\rangle$ = $\langle$*translation string*$\rangle$;

MathType equations (just like TEX ones) are represented internally as a tree. Let's take the following equation as an example:

$$y = \frac{a+b}{c}$$

MathType sees this equation as:

```
eqn (root)
  slot (main)
    character (y)
    character (=)
    template (fraction)
      slot (numerator)
        character (a)
        character (+)
        character (b)
      slot (denominator)
        character (c)
```

The translation process begins by applying the display equation rule,[2] to the root of the MathType expression:

```
eqn = "\[@n#@n\]@n";   // display equation
```

The characters between quotation marks are processed from left to right. Most of the characters in the `eqn` rule are simply placed in the output translation stream. The `@n` sequence outputs a newline. The `@` character, called the escape character, is used to insert special characters into the output stream. The default escape character is `$`, but is redefined in the TEX translators to `@` for convenience.

The `#` in the `eqn` rule causes the translator to look for a rule that will be used to translate the equation's main slot. After applying this translation (we'll get to that next) and inserting its output into the translation stream, the rest of the `eqn` translation string is output and the translation process is complete.

Let's go back to see how the `#` in the `eqn` rule is processed. This is done with the rule:

```
slot/t = "#";
```

This rule works just like the `eqn` rule but is even simpler. The `/t` option is used to signal that this

[2] The translation rules used in our example are simplified somewhat for the purpose of this paper.

rule is to be used for the top-most slot in the equation only. Other `slot` rules enclose the translation of their contents in {}, the TEX notation for grouping. Now, each item in the main slot is processed. The rules for `y` and `=` are also very simple:

```
char/0x0079 = "y";    // Latin small letter y
char/0x003D = " = ";  // Equals sign
```

Here is where Unicode comes into play. MathType knows the Unicode value for `y` is 79 (in hexadecimal notation). It uses this knowledge to find the `char` rule that specifies how `y` is to be translated.

The fraction is handled by a template rule:

```
frac = "\frac{#1}{#2}";  // fraction
```

The `#`s in this rule are followed by numerals that specify the slot's index in the template; `1` for the numerator, `2` for the denominator. These two slots are processed much like the main slot, except they use the more general slot rule:

```
slot = "{#}";
```

So, the complete translation of our simple example is:

```
\[
y = \frac{a+b}{c}
\]
```

MathML

The MathML specification was written by the W3C Math Working Group.[3] In April 1998, it was raised to Recommendation status by the W3C. MathML has as its main goals:

- encode mathematical material suitable for teaching and scientific communication at all levels
- encode both mathematical notation and mathematical meaning

MathML is intended to be used to both present mathematical notation and to serve as as a medium of exchange between scientific and mathematical software. Toward that end, MathML defines a set of XML elements and attributes (together called markup) that fall into two categories: presentation markup and content markup. Presentation markup is intended to describe mathematical expressions from a two-dimensional layout point of view, whereas content markup is intended to capture the meaning of the mathematics.

MathType provides four MathML translators (why there are four will be explained shortly) that can convert its equations into MathML's presentation markup. For example, translating the following expression into MathML:

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

results in:

```
<math displaystyle='true'>
   <mrow>
      <mfrac>
         <mrow>
            <mo>-</mo>
            <mi>b</mi><mo>&PlusMinus;</mo>
            <msqrt>
               <mrow>
                  <msup>
                     <mi>b</mi>
                     <mn>2</mn>
                  </msup>
                  <mo>-</mo>
                  <mn>4</mn><mi>a</mi><mi>c</mi>
               </mrow>
            </msqrt>
         </mrow>
         <mrow>
            <mn>2</mn><mi>a</mi>
         </mrow>
      </mfrac>
   </mrow>
</math>
```

The first reaction of most TEX users is a gasp at how verbose MathML is. Please bear in mind that MathML is not intended to be authored directly by humans but with tools like MathType. MathML inherits its verbosity from XML. This "disadvantage" is far outweighed by the advantages gained with XML structure with its support in browsers, editors, and other tools. In the future, we hope to see MathML become the language of choice for exchanging mathematics between mathematical applications. Its eventual integration with browsers should make it tremendously useful in teaching.

Unfortunately, it may be a little while before MathML achieves its promise. Until we are able to properly display MathML in the popular browsers, such as Microsoft's Internet Explorer and Netscape's Navigator, we will have to rely on various browser "plug-ins", such as IBM's techexplorer.[4] and Geometry Technologies' WebEQ.[5] These work but are constrained by various font issues, sizing problems, and lack baseline alignment for in-line math. The W3C's

[3] See `http://www.w3.org/Math/` for the specification and other information on MathML.

[4] For information, see `http://www.software.ibm.com/network/techexplorer/`.

[5] See `http://www.webeq.com/`.

Math Working Group has made browser integration one of its highest priorities.

In order to work with the various MathML browser plug-ins, we have had to create several versions of our MathML translator, one for each. They differ only in the "wrapper" code they place around the generated MathML in order to invoke the plug-in and pass the MathML code to it. When true browser integration is possible, we will only need a single MathML translator.

Experimenting with MathType's translators

Because of their simple and open structure, MathType's translators can be easily modified or new ones written from scratch. Some possibilities include:

- creating a new translator for the mathematics portion of an SGML document standard (DTD)
- changing an existing TEX translator to make use of some macro package or the author's own preferred macros
- using the Unicode capability of MathType's translators to take advantage of the various TEX adaptations for non-Ennglish languages

Document conversion

MathType's translation facilities can also be incorporated into the process of converting entire documents to TEX or any other language supported by one of its translators. MathType has a programmatic interface beside its more familiar graphical user interface. This interface is via functions in a Windows DLL (Dynamically Linked Library). As MathType is often used with Microsoft Word, we have provided functionality that can be accessed using commands on a "MathType" menu within the Word application itself. One of these is a Convert Equations command that can be used to convert all the MathType and Equation Editor equations in a document to any one of the languages for which a MathType translator is available. Although this does not result in a fully translated Word document, the most difficult part, converting equations, has been achieved.

MathType's Word support is written in that product's Visual Basic language. The source code is accessible and may be used as the basis for your own conversion scheme.

Conclusions

MathType 4.0, with its new translation features, can be used in a variety of ways:

- as an interactive front-end to TEX authoring
- as an aid to learning TEX
- to experiment with the new MathML standard
- with the development of custom translators, it can be used to generate virtually any text-based math language

It is our hope that TEX users will want to add it to their arsenal of useful tools.

TEX musings

Musings from the Bard

Oh, what a tangled web is TEX,
or so it seems at the outset;
for highest quality, the best to look,
Oh why did I choose to typeset my own book!
. . .
For Macintosh users the skies are quite blue,
with Barry to help you and Ben Salzburg too.
With Art, Ross, and Uwe ready to assist,
just send a short email to Gary Gray's (Textures) list.
. . .
If shareware type software is more to your taste,
your super-fast Power-Mac need not go to waste.
There's CMac- and Direct-TEX to lessen the sorrow,
and that great program OzTEX, by Andrew Trevorrow.
. . .
For Unix-like platforms the software's all free,
with a teTEX installation from the TEX-Live CD,
which collects all the pieces and orders all parts,
Thanks to Thomas Esser, Kaja, and Sebastian Rahtz.
. . .
Leslie Lamport created LATEX nearly fifteen years ago.
It evolved into 2-epsilon by a process rather slow.
Improvements are numerous; results you can see.
Thanks to Carlisle, Mittelbach and Rowley,
It'll be even better with release LATEX3.
. . .
For PostScript Type1 fonts exquisitely drawn,
The expert is Berthold, whose surname is Horn.
Where the yin meets the yang in the great cosmic goo,
Don't get this name mixed-up with Louis Vosloo.

—Ross Moore

Models and Languages for Formatted Documents

Chris Rowley
The Open University
527 Finchley Road
London NW3 7BG
United Kingdom
C.A.Rowley@open.ac.uk

Abstract

The largest change that has come to the world of document formatting since TEX's DVI language was designed is the need to support documents destined for multiple uses, e.g., for interactive reading on screen and for paper output. It is time to investigate what is needed, both now and in the immediate future, from a device-independent description language for formatted documents.

This paper does not provide a complete such investigation. Instead, it outlines what is required from a language that can describe the "device independent" properties of the "formatted form" of the currently imaginable range of documents. The important and innovative concept identified here is the complete integration of three central formatting aspects of on-screen documents: text, graphics and the mechanics of interaction.

Introduction

Motivation. There is currently a need for a standardised but flexible and comprehensive language that will enable the description of all aspects of formatted documents that are the output of high-quality document formatters such as the growing number based on TEX (and its derivatives); this would also then be available for the output of all the superior XML/XSL-based formatting software that, we are assured by those who control the world, will soon decorate our desk-tops.

Many aspects of formatted documents, such as graphics and colour, were deliberately excluded by TEX's designers but the largest change that has come to the world of document formatting *since* TEX's DVI language was designed is the need to support documents that are designed (to high standards) for multiple uses, e.g., for interactive reading on screen and for paper output. Many people now have experience of using TEX as the typesetting engine for such documents, producing as the multi-use output form a 'PDF document'. This could be either by use of pdfTEX or by producing PostScript and then converting this to PDF (the acronym PDF here refers to Adobe's Portable Document Format). The power of combining the programmability of TEX with a thorough knowledge of the capabilities of PDF viewers and Java programs have been brilliantly illustrated by Hans Hagen.

There probably also exist other necessary extensions to the currently used models for formatted documents that are not supported well by any current such language. Thus it is time to investigate what is needed, both now and in the immediate future, from a device-independent description language for formatted documents. In order to illustrate and crystallise these ideas, it is useful to consider how these needs are met by the current version of PDF or by other languages such as the Scalable Vector Graphics language. This would lead to a far longer paper detailing the achievements and failings of the current version of PDF and the relevance to this subject of the current thinking on SVG but here I have confined myself to occasional comments on pertinent aspects of these languages.

Background. About a year ago I was bold enough to state:

> By August 1999 I hope, with a bit of help from my friends, to have further analysed the models and concepts that need to be supported by a language for describing multi-use documents, and how well PDF provides such support.

Well, I got a lot of help, mostly from the PDF discussion list [8] and particularly from Sebastian Rahtz and Hans Hagen who, being privy to Adobe's future plans and hence in their thrall, could often only answer with the phrase "but I could not possibly

comment", even about things that are already described in the literature — such is the way of commerce.

The result is this short paper outlining what is required from a language that can describe the "device independent" properties of the "formatted form" of the currently imaginable range of documents. The discussion here tries to be general but it is heavily influenced by the currently popular resources in this area: DVI [4], PDF [2, 3] (and hence PostScript [1]), together with some, such as the Scalable Vector Graphics (SVG) Specification [5], that are currently under development.

I am very much aware that the current version of this paper lacks a lot of explanation and examples; and that it contains little about practical ways to take these ideas forward and relate them to other activity in this area. However, it does contain at least one significant new idea: the complete integration of three central formatting aspects of on-screen documents: text, graphics and the mechanics of interaction; this will lead to a more comprehensible and systematic treatment of all aspects of multi-use formatted documents.

Preamble. I shall assume that the reader has some familiarity with the DVI language (at least the commonly used parts) and with the PDF language (v1.2 or later, but only the formatting-related parts).

Please note that there are many things that are not covered in this paper because, although very important for modern document science, they are not directly relevant to the current subject. For example, since we are considering a description language for formatted, multi-use documents, we completely ignore the current uses, aimed at expressing documents as logical tree-structures, of languages such as XML and HTML (although these are often used to provide an inspired mixture of semi-specified formatting and logical markup). It is also possible to combine such languages with a language such as PDF to describe "partially formatted" documents.

The major consequences of the chosen language for the design of the applications, e.g., TEX or its successors, that produce examples of it will be mentioned, but only in passing and hence incompletely. However, these are probably of greater practical importance than the details of the language itself.

The paper begins by setting up the context, describing briefly the relationship between DVI, PDF and the various models of document formatting into which they fit. It then describes various models that must be supported by a fully functional language for multi-use formatted documents and analyses the consequences of these for the structure of the language.

Subsequent work will consider in more detail the specification of the language together with the design and implementation of related applications.

DVI and PDF. One motivation for this paper was my being asked at the TUG'98 conference: which is better, DVI or PDF? My reaction then was: since they are so similar, neither! But then I was thinking only of the original version of PDF; now, having discovered the joys of v1.2 and more recently the 7.4MB of the latest (v1.3) manual, I publicly recant from that position.

Although PDF is technically not a "device independent" language, it contains a large core of stuff that is, at least potentially, as "device independent" as TEX's eponymous DVI language. Both must, of course, be parsed by an application that understands the language and its underlying document model, formatting model and page model; and, although they look very different at the detailed level, the page models of these two languages (and their abstract semantics) also have a lot in common. This is one reason why the part of pdfTEX [6] that handles classical TEX files is only very locally and minimally different from classic TEX. However, PDF has a somewhat richer document model and it integrates text and graphics in its formatting model. This is one reason why pdfTEX has extra primitives.

On the other hand, PDF also has a large, and growing, part that is dependent on the very specific, and limited, features of Adobe's own viewers and font technologies. As with most languages that are being actively developed whilst being widely used, PDF is now a mixture of good and bad ideas: it is still based on some simple but general models but these are not always used to provide extensions nor have they been developed to provide more inclusive but equally clean new models. Instead, it has grown a collection of ad hoc add-ons that lack simplicity and coherence. Much of the additional functionality of pdfTEX is there only to support such very particular features of PDF, as its meta-data objects and stream compression possibilities.

Although this is a lot, PDF does only what it does; it has become a more difficult language for a (human) document formatter or programmer to work with. It is therefore currently not at all clear how to make straightforward adaptations or extensions of the PDF language and we seem able to get from it only what They want us to have in Our documents. Note that some caution is needed when evaluating the PDF language itself since much of its

expressive ability is obscured by the lack of functionality and bad behaviour of the applications currently available for viewing or translating it.

Given the close symbiosis between its development and that of the Adobe's Acrobat Reader application, it is very likely that, wisely used, it is a very good language for rapid and accurate screening of downloaded documents. However, this is by no means a unanimous verdict on the utility of PDF and any such advantages have clearly been at the expense of efficient and accurate production of those documents since the current version is far from providing the uniform, clean, comprehensible interfaces needed by writers of applications.

This suggests that there is a need for Yet Another Language: one with a clean and general model and a flexible, uniform syntax. If this is incompatible with fast incremental processing then a compilation process should be inserted to transform this into something at least as good as PDF. So here are some thoughts on such a language.

Background and models

Here is a description of the use of a FDL (or Fixed Formatted Document Language) and the models of document processing that it both supports and provides.

The global model. The assumed usage of this FDL derives from the following high-level model of the document formatting process in which it is used.

- A *generating application* (GA) produces a fully formatted document and outputs it in this FDL language.
- A *processing application* (PA) uses the information about a fully formatted document described in this FDL in order to do only the following (these are informal descriptions):
  - faithfully render (in accordance with the medium) parts of the visual content of the document on one or more media;
  - when appropriate, supply information about attributes of such a rendering needed to determine the interaction state of the PA;
  - pass on, but not process, information streams to other applications (in particular, non-visual material).

At its top level, a formatted document consists of a collection of objects, the most pertinent of which are *formatted objects* (FOs).

A model for the language. What information must therefore definitely be represented in an FDL description of a document? Here is an answer.

- The contents of the document that are needed for rendering the FOs in the document on any supported output medium.
- The contents that are needed to determine the interaction state on any supported interactive output medium.
- Information about structural relationships amongst the formatted objects in the document.
- Pointers to other resources required for the rendering process (e.g., rasterisation, font and colour information).

The following is information that is not essential but is useful; it is also very closely related to the formatted document. Other (non-formatted) objects contain such information.

- Information about the logical structure of the document and its relationship to the structure of the formatted document.
- Information needed for non-rendering activities for which support is needed; in general, this is too open-ended but some of these are the logical information that is needed for activities that are traditionally associated with on-line document readers, such as indexing and searching.

There are, of course, many other things that are essential to the complete description of a document and it may well be appropriate to add to the language objects to be used for their specification. The following are some examples (from many) of information that is important to the document but is not, per se, closely related to the formatted form of the document.

- Database information about the document itself rather than its contents.
- How any visual material produced by other cooperating applications (which may not themselves process the FDL material in this document) should be placed relative to the rendering of the document.

This paper will thus analyse in detail only the information in the first group (of four items). It will also discuss some ideas concerning the information in the second group but will argue that the FDL needs to be able to express only how the provision of such information relates to information in the first group, leaving the specification of most of this information to other, more suitable, languages; these languages have been, or will be, developed elsewhere and can be used in a wider context.

The FDL is not intended to provide a revisable document format. Thus it will contain no provision below the level of the FOs for the specification of

user-level graphical objects so that they can be directly manipulated, as in a drawing application or a document editor. This should not rule out support for extensions that, like PDF, provide some very limited but useful form of structured revisions of FDL documents; however, this is not a primary property of the FDL.

Further restrictions. In order to appease the editor of these proceedings and to put a reasonable limit on the time I spend writing, I shall here restrict the analysis and discussion in the following ways.

- The top-level formatted object (FO) described by the FDL will be a two-dimensional, unrotatable rectangle (this is a convenient but not essential restriction).
- The graphical model will have no concept of transparency, i.e., no graphical layers: this reflects only the current limit on my resources for investigating the issues involved and should be relaxed as soon as possible. It is also one of the areas where PDF's support falls short of current requirements.
- There is no concept of time nor of an external environment beyond the idealised two-dimensional output medium; hence the FDL itself does not describe the non-typographic content of sound/video; and documents cannot be defined to look different on Wednesdays, on Macs or on Vancouver Island (although some such requirements could, of course, be implemented by the PA).
- There is no concept of service levels to be negotiated between a client, knowing its local PA resources, and a document server.

Note that the first restriction does not limit the scope for specifying what is displayed since, within these top-level FOs, complex clipping paths can be specified.

Note also that the PA can use information expressed in the FDL to do complex things such as affording different views of the document and controlling time-dependent actions to produce *son-et-lumière* shows, etc., but these do not need to be described directly within the FDL.

Moreover, the information needed to control associated multi-media actions should be encoded in languages designed explicitly for describing such objects and these languages should not be part of the FDL.

A model for the medium. The abstract model of the visual medium is therefore a rectangular subset of a mathematical Euclidean plane on which are defined attribute functions such as "colour". Thus other technical issues not dealt with here are the precision of numerical values and the closely related provision of rasterisation information. These are very important in practice but it is best to keep them clearly separate from the raster-independent, arbitrary precision part of the model. In addition (or rather subtraction) many of the complexities of colour and tone rendering are not present in the model since these are intimately connected to the rasterisation process.

Having so peremptorily dismissed rasterisation from this formal model, I must quickly explain that everything in the model is predicated on a model of device-dependent *rendering* that involves a rasterisation of this idealised plane.

Analogies. One can liken a simple implementation of this global model to a translator (the GA) and its agent (the PA), where: the translator compiles application-oriented document formats into a well-defined, machine-oriented representation of the visual form of the document; the agent processes this lower-level code. In this simpler paradigm, the "model for the language" is analogous to the operational semantics of that machine-oriented representation and the "model for the medium" would be the abstract architecture of the machine.

A heuristically better, but less precise, analogue is with database models that include pre-compiled views and data indexes.

Analysis

At the lowest level such an FDL needs to be able to express, within the above limitations, full details of the following, and nothing more:

- everything that could be visually displayed by any PA using any supported visual medium;
- everything that is needed for the detection of interaction events by any PA that supports interactivity with such a visual mediuml

This information can be usefully divided into a number of related topics but they are all, ultimately, graphical abstractions.

Graphical specifications. Here we separate the concept of text (i.e., glyphs from fonts) from other visual items; however, we do not separate the interaction-related graphical information from the visual parts.

The underlying model for all this information is the specification of regions in the visual medium (idealised as a mathematical plane). These are the only fundamental graphical objects that are used.

Although there are many other possibilities, there is no clear reason to depart from, or extend, the commonly used collection of methods for specifying regions in terms of cubic paths; see, for example, the paper version of the original PDF specification [2]. This almost universal method is also used by PostScript and SVG.

Paths. These are used solely as a way of defining regions (stroking, etc., define a narrow area along the path). They are typically piece-wise cubics, other common forms such as conics being provided by the language only as syntax for their cubic approximations.

Note that these paths are mathematical idealisations that are then used to define the somewhat more concrete regions by means of various operators such as clipping, stroking and filling. Note that the use of these words here does not imply that any painting of the defined region is yet specified.

Regions. Having thus defined a region, it can be (abstractly) painted in some way or it can be given a label for use in defining interaction events; these are not exclusive possibilities. Note that the specification of the region is identical for both visual information and for these interaction labels. This is all that is needed from the FDL in order to support all currently used types of interaction. A region can both have a label and be painted, and these two properties are completely independent. It may be sensible for all regions to be labelled objects so that all their properties, including painting related operations, would be accessed via the label.

Events. The first stage is to define events; although there is a good case for a generic language to be used here, the present level of development of the technology suggests that ad hoc languages closely linked to particular devices may be needed for some time. That used by SVG is a good example of such limited expressibility.

Thus the FDL needs *event definition objects* where the following information can be put, using a suitable language:

- definitions of the interaction events,
- the actions associated with interaction events.

An important and common case of an action is to display a particular view of the current, or some other, FDL document; the features needed to support this are described below.

Painting. Much of what comes under the detailed specification of a painting method is relevant only to the details of the rasterisation but some, such as colour information, also need methods for device-independent specification. The most general colour information is the specification of a colour gradient function, to specify how a region should be painted; this is a mapping from the abstract visual medium to a colour space. There is a need for further investigation into what types of mappings are needed here; SVG will support a small range of mappings, including linear, radial and periodic (for patterns).

This could be extended to support the far more general concept of getting such resources from an external paint server (not to be confused with the Mix-Yer-Own machine outside the local Do-It-Yourself store); this is analogous to the commonly used indirect ways of specifying glyphs and other font resources.

Text. Although glyphs are also graphical objects, the methods by which they are specified are typically so completely different that treating the two similarly becomes fatuous. In particular, the choice and positioning of glyphs typically requires external resources and, hence, other languages. In the case of PDF and PostScript, this is the only supported underlying model for text: both positioning and rendering information for typical fonts can only be specified via a fixed external font resource that must be accessed via a fixed-size encoding table. Such external font resources are used by a specialised glyph-rendering part of the PA and also, often, by the GA: it is clearly essential, but often difficult to achieve, that these two applications use identical information.

Whilst there are good reasons to support most of these existing models and formats for glyph production, the FDL must support a far wider range allowing, if feasible, for future new glyph resources and font technologies as they come into use. Thus it should support the specification of all of the following:

- font-resource independent specification of a glyph within a font;
- explicit positioning of glyphs;
- relative positioning of sequences of glyphs (using font resources to calculate exact positioning): at least for all standard typesetting modes, both horizontal and vertical, possibly also for typesetting along more general graphics paths.

Although perhaps not strictly part of the FDL itself, a clear requirement arising from these is the ability to attach arbitrary external resources to a FDL file.

Higher-level structure. The basic formatted objects (FOs) can be related in various ways, including these three of immediate importance:

- *Logical arrangements*: these can be very general relationships but include traditional page sequences; these do not prescribe anything about the formatting of the individual objects.
- *Formatted arrangements*: use and reuse of objects within others.
- *Global information* that allows viewers/printers to define different views of a document in terms of these objects: e.g., print sequences, relative positioning of windows on a screen, suppressing the rendering of the content of whole objects (another area where PDF is currently deficient).

I see no reason to put any restrictions within the language on the nature of these relationships, thus at least a *general labelled-graph* language providing arbitrary linking information is needed here.

Such relationships and their specification need further investigation and development. Specification of the formatting relationships will immediately require an extended model of the medium that supports layers and transparency.

There is currently some small-scale research activity concerned with logical information extensions to PDF, in particular the work on structured-PDF at Nottingham University. It is unclear whether Adobe have any long-term interest in moving PDF in that direction (or, indeed, whether they have any interest at all in the language itself as anything beyond a cryptic internal language for the Acrobat black-boxes).

Trade-offs. Many of the choices that need to be made in developing the detailed syntax of the language lead to decisions that, whilst not affecting the semantics or power of the language, do affect the following measures of its utility. The first two items in this list are independent of any particular document, whereas the others will vary according to the type of document and its uses:

1. the expected functionality of the GA,
2. the required functionality of the PA,
3. the relative size of the FDL file,
4. the relative speed of the generation of the FDL file,
5. the relative speed of accessing information in the FDL file,
6. the relative speed of processing information from the FDL file.

In general, decreasing 1, and hence, typically, 4, will increase 2 and, often, also 3, 5 and 6. For example, if the FDL supports a large range of higher-level graphical objects, such as transformations, arcs of conics or smooth piecewise-cubic paths, then the GA does not have to be able to turn these into basic cubic paths but the PA must be able to process them.

Of course, increasing the amount of information (e.g., font resources) that does not need to be stored in the FDL file also decreases 3, but it also requires the PA to be able to access these resources effectively.

This section does not analyse the possibilities for the use of alternative formats since these affect equally any language. Some relevant techniques are data compression, which is comprehensively supported by the PDF standard, and binary formats that can be read quickly, as typically used by DVI but not currently available in PDF.

Summary

Outline. A formatted document, as described by an FDL specification, is a collection of reusable FOs with labelled relationships. These FOs contain positioned graphical objects including, recursively, further FOs; but they have no further internal structure. No distinction is made between the graphical objects used for painting and those used to define interaction events.

Interaction events and associated actions are not described in the FDL itself but it provides objects specifically to contain these descriptions. It also provides objects for describing external resources and the possibility to attach such resources to an FDL file.

Although glyphs are graphical objects, they are most often accessed via external resources so they must be treated very differently within the FDL.

All the organisational structure of the formatted document is defined in the FDL by general named relationships between the FOs; other logical information is not described in the FDL itself.

General principles. In developing the details of such a language the following principles should be adhered to as much as possible.

- *Indirection*: always A Good Thing.
- *Modularity*: but do not try to separate too rashly things that should be intimately connected.
- *Flexibility*: do not impose unnecessary restrictions on the GAs or PAs.
- *Extensibility*: of course! But only within the limits of the above outline.
- *Clarity*: and ease-of-use as the cream on the cake!

The way forward

The next step is to refine and formalise the ideas described here and to investigate extensions of these

models that will support, in particular, a powerful concept of layers in the output medium.

Some possible (and mutually supportive) ways in which the TEX community should be able to assist in this, and beyond, are as follows:

- Further extend DVI along the lines suggested by the NTG TEX Future Working Group [7]. This already supplies a syntax for those graphics objects supported by PDF; semantics satisfying the FDL model can be simply specified for these and further necessary objects.
- Work with the W3C group to influence the development of the SVG specification so that it can be used as (part of) an FDL.
- Design future typesetting systems (such as NTS) to output such a powerful, clean FDL and write drivers that use it directly or translate it to a more processor-friendly and currently supported language, such as PDF or the API (A.. P.. I..) of a printer/viewer sub-system.

Or maybe I should move on to even more radical ideas whilst PDF and SVG/XML slog it out in the market place and TEX/DVI continues to dominate the quality niche? (Note: The last word must be treated *à la française* to make the pun work.)

Postamble

Preparing the conference talk and the subsequent discussions have shown that there are other important questions not even touched on above; thus, this paper should be treated as "preliminary thoughts".

In particular, it became very clear when I was designing and producing the examples for the talk that, even by using all currently available applications (including some pre-release versions), I could not implement all of those features of a formatted document that are currently agreed to be desirable. Moreover, it has taught me that, even at the high level of my models and languages, there is a lot more to the interactions amongst graphics, text, and screen formatting than I had considered so far.

For example, what should happen when a user resizes a window that contains both graphics and text, possibly intimately connected? The possibilities for each element are as follows (at least): resize, clip, reflow.

Resizing may make sense, within reasonable limits, for some graphics but maybe not for others; it is rarely the best thing for text. Contrariwise, reflowing is not usually feasible for graphics but may be sensible for text, again within some limits.

So who decides what is allowed? The author should at least be able to define the reasonable limits but maybe the user should have some control over what he is looking at.

Thus, more work, more ideas and, sadly, more papers, are needed.

References

[1] *PostScript Language Reference*, 3rd ed., Adobe Systems, 1999. `http://www.adobe.com/prodindex/postscript/`.

[2] *Portable Document Format Reference Manual*, v 1.2. Adobe Systems, 1996.

[3] *Portable Document Format Reference Manual*, v 1.3. Adobe Systems, 1999. `http://www.pdfzone.com/resources/`.

[4] *The DVIType Program*. Stanford University, 1982. `http://www.CTAN.org/tex-archive/systems/knuth/texware/`.

[5] *Scalable Vector Graphics (SVG) Specification*. W3C, 1999. `http://www.w3.org/TR/WD-SVG/`.

[6] *The pdfTEX Manual*. 1999. `http://www.tug.org/applications/pdftex/`.

[7] NTG TEX future working group. TEX in 2003: Part II: Proposal for `\special` standard. *TUGboat* 19(3) pages 330–337, 1998.

[8] PDF e-mail discussion list. `http://tug.org/mail-archives/pdftex/`.

[9] Hagen, Hans. Examples of the use of pdfTEX to produce interactive documents. 1999. `http://www.pragma-ade.nl/`.

AcroTEX: Acrobat and TEX Team Up

D.P. Story
Department of Mathematics and Computer Science
The University of Akron
Akron, OH 44325
dpstory@uakron.edu
http://www.math.uakron.edu/~dpstory/

Abstract

Adobe's Acrobat (PDF) and Donald Knuth's TEX system make a powerful team for putting mathematics on the internet. For the educator, this team, called "AcroTEX," is the poor man's multimedia software company.

Though TEX was implemented before the rise in popularity of the World Wide Web, PostScript code written to the `.dvi` file, using TEX's `\special`s, can be used to enhance an electronic document created from a TEX source by introducing such elements as color, hypertext links, form features, sounds, and even video clips. These special features are achieved by inserting 'pdfmarks' into the output file. The Adobe Distiller, in turn, interprets these pdfmarks and translates them into the appropriate element as it writes the PDF document. TEX is, therefore, well suited for creating PDF files, especially technical material. With the aid of its very powerful macro facility and the ability to position material very precisely on a page, these electronic enhancements can be created and placed in an exact and automated way.

This paper explores the capabilities of AcroTEX and the contents of the AcroTEX web site (`www.math.uakron.edu/~dpstory/acrotex.html`); examples include tutorials, an electronic grading system, mathematical games, and technical articles.

Introduction

In this paper, I will describe how TEX, the outstanding mathematical typesetting engine, and the Portable Document Format (PDF) of Adobe Systems, first introduced in the Acrobat suite of software, form a team called "team AcroTEX",[1] and how this team has opened up a world of possibilities to people who are interested in electronic education.

From the perspective of an educator, this paper is an exposition of the natural *implications* of combining TEX and PDF; the exposition covers genesis (first thoughts), creation (of tutorials), problems and solutions, educational games, technical articles, and new macro packages for readers who may want to develop on-line educational materials themselves.

Here, AcroTEX refers to a process of producing PDF documents from a TEX source. A PDF document is a compact, self-contained file format which preserves the page layout of the authoring application. This makes a PDF file suitable for distribution over the Web.

AcroTEX also refers to a web site:

`www.math.uakron.edu/~dpstory/acrotex.html`

The documents referenced in this paper can be accessed at the AcroTEX web site, a site primarily dedicated to mathematical education. All files at this site, save only some start-up HTML pages, are in PDF format.[2]

Genesis

The original concept was to create a series of electronic tutorials covering some of the topics of the first semester of a standard course of calculus as offered in many colleges and universities in the U.S. The design goals of the tutorials included typeset quality on-screen mathematics, cross-referencing using hypertext links, and on-screen color.

Typeset quality mathematics and a presence of limited finances implies TEX. Of the freeware and

[1] A proposed alternative is TEXrobat, but this sounds too mechanical.

[2] Readers are available for virtually every platform; see `www.adobe.com/prodindex/acrobat/readstep.html`.

commercial TEX systems available at the time, late 1994, only the Y&Y TEX system offered coloring of fonts and a hypertext feature within its .dvi previewer, DVIWindo. A small site license from Y&Y was purchased using a start-up grant from the Educational Research and Development Office at the University of Akron.

The calculus tutorial, called "e-Calculus", was written and put on the department intranet as a collection of .dvi files. The students would come into the computer lab to review the materials using Y&Y's DVIWindo. The system worked well; however, the students were reluctant to spend the long periods of time that would be necessary to read the tutorial in the computer lab.

This reluctance prompted me to consider the internet. For many reasons, the .dvi file format is not a suitable format for the tutorials on the Web. Adobe's PDF seemed to be a natural choice: the typeset quality of the material, page layout, and color all would be preserved.

Fortunately, Y&Y was well positioned to convert its .dvi files to .pdf files. Their dvi-to-ps driver, dvipsone, automatically converted the hypertext links that DVIWindo understood to hypertext links that the Acrobat Reader understood. As a result, "e-Calculus" was ported to the internet with very little trouble.[3] This was the beginning of the AcroTEX web site.

The site has since grown in size to include other tutorials, mathematical games, a demonstration of forms processing using the PDF forms format FDF, several technical articles on TEX, LATEX and PDF, and a couple of macro packages (web/exerquiz).

The tutorials

The writing of the "e-Calculus" tutorial was simple enough and will not be discussed here; another tutorial, entitled "An Algebra Review in Ten Lessons" (mpt_home.html), has since been written in response to the needs of the incoming students at the University of Akron.

Beyond the technical discussions of calculus or algebra that any paper document would provide — though the discussions are more verbose than would normally be seen on paper — these tutorials try to take advantage of the electronic medium.

For finding information within the tutorials, hypertext versions of tables of contents, bookmarks (a feature of PDF), cross-references, and indexes — both local (for the file being viewed) and global (for the entire calculus/algebra set of tutorials) — are all provided.

The tutorials include in-line examples and exercises *à la Knuth*: that is, in the source file, the questions and solutions are kept together; however, solutions are linked to the questions by hypertext. The syntax is as follows:

```
\exercise < Some exercise question. >
\answer < The answer or solution. >
  ⋮    ⋮    ⋮    ⋮    ⋮    ⋮    ⋮
\endexercise
```

The macro \exercise sets the hypertext link, \answer sets the target, a named destination, for the link and starts a verbatim write macro. All material between \answer and \endexercise is written to the file \jobname.ans, which is later included at the end of the main file. There is an \example macro that behaves in the same way to handle examples.

This method of handling exercises and examples allows the posing of the question within the body of the text, but the solution does not appear (and take up screen space) unless the reader wants to see it. This allows for a clean, clear, more orderly presentation and discussion of topics.

By the way, proofs of theorems are handled the same way. The theorem is stated and a hypertext link is given to the proof. The main text then continues with a discussion of the meaning of the theorem, followed by examples and exercises.

Another feature of the tutorials is user interaction. User interaction with the document comes in the form of multiple-choice questions or quizzes. Click on the chosen response to obtain instant feedback in the form of humorous congratulations — or an error message.

TEX and PDF

Macro packages. When I first started this project in 1994, I decided to use $\mathcal{AMS}$-TEX 2.1 as the basic macro package. At the time, I had a rather slow computer with very little RAM. I found that LATEX was very slow in loading and took a long time to process a file; however, $\mathcal{AMS}$-TEX 2.1 on the same system loaded quickly and ran acceptably fast. The consequences of that decision meant: (1) I must write all my own macros for page formatting, cross-referencing, tables of contents, indexes, color inclusion, hypertext linking, etc., and (2) I must read *The TEXbook* not once, not twice, but many times. Other books studied and found to be very useful are the ones by Salomon (1995) and Eijkhout (1992).

[3] File: e-calculus.html. All files mentioned in this paper are located at www.math.uakron.edu/~dpstory.

If I were starting over today, I would probably use LaTeX and Sebastian Rahtz' `hyperref` package. LaTeX comes with many of the features that I had to struggle with to include in my own system; `hyperref` provides many of the hypertext features that I regularly use in my tutorials.

However, I have no regrets. At the time I began this project in late 1994, `hyperref` was still in its infancy anyway. One thing is certain, since I wrote the macros myself, *if something goes wrong, I know immediately where the problem is, and how to fix it.* Problem turnaround time is much shorter: I don't have to complain (report bugs) to the package author, then wait for the fix to arrive.

Creating PDF. There are essentially three methods for creating an *interactive* PDF document from a TeX source:

1. Convert the `.dvi` output file to a PostScript file using a `dvi`-to-PostScript driver such as dvipsone or dvips, then use the Acrobat Distiller to convert the PostScript file to a PDF file.[4]
2. Convert the `.dvi` file to PDF, bypassing the PostScript step, by using either dvipdf (Lesenko, 1996) or dvipdfm[5] by Mark A. Wicks.
3. Convert the `.tex` source file directly to a PDF file by using PDFTeX,[6] a program that is the ongoing labor of Hàn Thế Thành (Sojka, Thanh, and Zlatuška, 1996).

Figure 1 depicts the various paths from a `.tex` source file to a `.pdf` file.

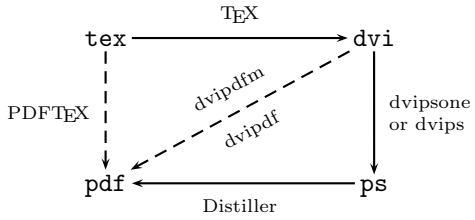


Figure 1: From TeX to PDF

Use Type 1 fonts. One last point must be made before leaving this topic. To create a *quality* PDF document from a TeX source it is necessary to use Type 1 fonts. The traditional font used by many freeware TeX systems is the bitmap or `pk` font; these fonts look choppy and jagged when incorporated into a PDF document and viewed on screen (though they do print decently).

Quality Type 1 CM fonts have been made available by a consortium of Bluesky Research, Y&Y, AMS, SIAM, IBM, and Elsevier. The freeware, shareware, and commercial TeX systems now come with Type 1 fonts. For an author wanting to publish on the Web using PDF, every effort must be made to reconfigure their TeX system to use these quality fonts.

Problems with multi-file systems

In this section, problems and issues associated with maintaining a large multi-file system are discussed. Hopefully, there will be enough detail to help readers better manage their own systems.

The use of a Make utility. The tutorials consist of a large number of files that are undergoing constant revision. A make utility[7] is used to help maintain this system of files.

A script file listing file dependencies was developed for each of the two tutorials, "e-Calculus" and "Algebra Review". (Actually, two sets of scripts are maintained: one for the system of `.dvi` files and the other for the system of `.pdf` files.) The make utility reads the script and initiates the programed action, perhaps calling the TeX compiler or the `dvi`-to-`ps` converter.

To create PDF files, for example, the `make` utility, as signaled by the controlling script file, calls the `dvi`-to-`ps` converter (dvipsone in my case) for each `.dvi` that needs to be updated, which in turn dumps the PostScript output into "Watched Folders". The Acrobat Distiller is activated and distills the files in these watched folders automatically and places them in an `out` folder. A batch file is then invoked to move the new `.pdf` files into their proper location.

The behavior of the `make` can be controlled by way of command-line switches of the make utility. As a result, only the files that have changed can be updated or the whole system of files can be re-TeXed and (optionally) re-distilled.

The `make` utility has been very helpful with the problem of trying to keep all files up-to-date. Updating the whole system of files is a matter of starting the `make` utility twice, once to TeX all files, then again to create the corresponding PostScript files dropped into the watched folders. The distiller does the rest.

Macro option switches. Each file belonging to one of the tutorials contains a table of contents, an

[4] This method is the process referred to as 'AcroTeX'; it is the only one that uses the Acrobat Distiller.

[5] Information and download are available at `http://odo.kettering.edu/dvipdfm/`.

[6] See `www.tug.org/applications/pdftex/` for more information and download links.

[7] The make utility that came with Microsoft Macro Assembler 5.0 is used.

index, and cross-document hypertext jumps. To further complicate matters, a collection of files covering a common topic, such as "Integration", shares the same table of contents (and same collection of bookmarks). When a new section is introduced or a new cross-document link is created, the supporting auxiliary files (and there are eight of them) must be properly updated and synchronized.

LATEX has a more or less automated system of updating the auxiliary files. Usually, LATEXing three times does the job. However, the macros developed for the multi-file system of tutorials are not nearly as automated but still are quite effective.

The system of macros that I have developed has "option switches" to update any cross-references between files, update the tables of contents, or update the indexes.

Below is a verbatim listing of (a portion of) the preamble of one of my "e-Calculus" files.

```
\documentstyle{tutorial}
\LocalOptions={}
\InstallOptions
```

The `\LocalOptions` token list allows local control of what auxiliary information is written. For example, after some changes in the file, this option can be changed to read

```
\LocalOptions={\CompileTOC}
```

The updated table of contents information will now be written to the file `\jobname.toc`.

In addition to local control of a file, global control of a collection of files is needed as well. In this case, the master macro style file, `tutorial.sty`, is opened and a token list, `\GlobalOptions`, is modified.

For example, the procedure for updating the entire "e-Calculus" tutorial is as follows. First, all local options are turned off and a `\GlobalOptions` token list is modified to perform each of the following tasks in turn: write to the appropriate auxiliary file (1) all table of contents entries, (2) all cross-document labels, and (3) all index entries. The "e-Calculus" source files are then TEXed (with the aid of the make utility) with each of these three tasks set. Finally, MakeIndex is run, the system is re-TEXed again. The entire multi-file system of PDF files can now be uploaded to the AcroTEX web site.

Write the document state. The tutorial system consists of a series of related articles. Files are kept to a size of around 500K in order to minimize the download time yet maximize the content.

Sometimes a single article is spread over several files. In this case, it is desired to have correct numbering of sections, examples, exercises, figures, and so on. For this purpose I wrote a macro called `\WriteDocumentStateTo{<filename>}`. The macro is placed at the end of a file and its purpose is to write the values of several count registers to the file called `<filename>.sts`. Here is a sample listing of one of the `.sts` files:

```
\secno=2 \subsecno=6 \resultno=2 \exno=43
\exmplno=14 \tagno=11 \figureno=2
```

The file `<filename>.sts` is then input by the file `<filename>.tex`, which sets the ongoing count of each of the listed registers.

Notice that the count register `\pageno` is not transmitted to the next file. This is because, at the time the system was designed, page numbers were determined to be of little importance in a multi-file system of tutorials!

No page numbers. Of course, TEX and PDF both maintain physical page numbers. In the tutorials, they are not printed on the electronic page and, with one exception, not referred to at all.

Because the tutorials were designed specifically for on-screen reading — not for the printed page — all cross-references can be made using hypertext links to named destinations. There is no need to write "see the definition of continuity on page 106"; it suffices to write "see the definition of continuity", where continuity is a hypertext link. (In the tutorial, links are color-coded; for this paper publication, they are underlined but do not work.)

Multi-file indexes. That one exception is in the creation of indexes. Page numbers in the index are used only as a visual reference of how far an index entry lies in from the beginning of the file.

Another problem with indexes in a multi-file system is that a given keyword, say "Euclid", might be referenced in several different files — perhaps, in an article on functions, on limits, on continuity, on differentiation, or on integration. How can the index give an *hint as to the* context *of the reference*?

Looking at the index in "e-Calculus" you would see the following entry under "Euclid":

```
Euclid, c1l:12, c1i:205
```

Each entry has an *index descriptor* followed by a colon and a page number. The index descriptor describes the file the reference lies in; for example, `c1l:12` indicates that the word "Euclid" was used on page 12 in the calculus 1 file on limits (`c1l`). The page numbers are underlined and hypertext-linked to the indicated page in the appropriate file.

The technique of manipulating the MakeIndex utility to obtain this index descriptor prefixed to the

page numbers is a little sneaky and should, perhaps, be left for another occasion.

Mathematical games

Games stimulate interest in mathematics, promote creativity, and provide a resource of class projects. All of the games listed below suggest that TEX and PDF can be used to create simple games that are inexpensive to build and challenging, educational, and informative to play.

My interest in games was stimulated by Gary Cosimini of Adobe. He created a game board in PDF that fascinated me; I wondered how he did it and set out to duplicate and expand on his game. To construct my own version of the game from a TEX source, I determined that I needed a greater understanding of the `pdfmark` operator. My reading and experimentation resulted in a Jeopardy-like game called "Algeboard" covering some topics in elementary algebra.

Studying the *Pdfmark Reference Manual* (Bienz and Staas, 1996) and the *PDF Reference Manual* (Bienz et al., 1996) gave me greater understanding of how to construct the game. With TEX's macro feature and its ability to place objects very precisely, I was able to stack form fields one on top of the other, and control their hidden attributes. When the user clicks on one of the game squares, the visible form field becomes hidden and another becomes visible. This is the way some of the effects are accomplished in the game.

"The Giants of Calculus", a two-column game of matching, was another challenge in coordinating layered form fields to get the desired effects.

Later, when Adobe Acrobat 3.0 came out with JavaScript, the Algeboard game was revised and JavaScript was used to keep track of the score. Thus was born "Algeboard/JS". This gave me some early experience in using JavaScript.

The experience with JavaScript helped when I attempted to create the more complicated "PDF Flash Cards for Kids", an electronic variation on the flash cards children use to practice their arithmetic skills of addition, subtraction, multiplication, and division.

All of the above-mentioned games can be downloaded from the AcroTEX web site.

Technical documentation

One of the goals of the AcroTEX site is to try to encourage other educators to use TEX (or LATEX) and the Portable Document Format to put mathematics (or any technical material) that might be of benefit to students on the internet. To that end, I've written several technical articles that describe how to insert hypertext links and form annotations into a PDF document, and how to use LATEX to create a quality interactive document in PDF.

About `Pdfmarks`. A `pdfmark` operator is an extension to the PostScript language that is read and understood by the Acrobat Distiller. The `pdfmark` is used to insert PDF-related features such as hypertext and form annotations. Primary documentation on `pdfmark`s is from Bienz and Staas (1996); an excellent on-line reference to `pdfmark`s is the "pdfmark Primer", one of the chapters from Thomas Merz' fine book, *Web Publishing with Acrobat/PDF* (1998).

For the TEX programmer wanting to create hypertext links or to insert form elements into a document, the appropriate code can be inserted into the `.dvi` file by using raw PostScript `\special`s. This code is then passed on to the `.ps` file by `dvi`-to-`ps` converters such as dvipsone or dvips.

For LATEX users, Sebastian Rahtz' `hyperref` package performs all these tasks wonderfully. Still, there are certain special effects that `hyperref` does not include; in this case, knowledge of `pdfmark` programming is essential.

The electronic article, "Pdfmarks: Links and Forms" (Story, 1998a), was written not long after I had constructed the games just described. I had made an in-depth study of the `pdfmark` operator and thought it might be a useful to write about what I had learned as a way of organizing the information in my own mind.

The article describes the basic, the advanced, and the more subtle techniques of creating hypertext links and form annotations for PDF. Written from the perspective of a TEX user, the article is interactive and highly informative. The many extensive and detailed examples contained in the article use TEX primitives and macros; TEX users can copy and paste code swatches into their own source document.

Authoring PDF documents using LATEX. In the past few years, I've received several inquiries from educators about how to construct good online tutorials using LATEX. Not a regular user of LATEX myself, I really didn't know. Ultimately, I got interested in learning LATEX, and in understanding how to use Sebastian Rahtz' `hyperref` package. As a result, I wrote the article, "Using LATEX to Create Quality PDF Documents for the WWW" (Story,

1998b). In this article, I tried to describe all the elements that go into the creation of a visually attractive, interactive PDF document using LaTeX, with special emphasis on how to use the `hyperref` package.

The `Web` and `Exerquiz` packages

The goals of these two packages are: (1) to create an attractive, easy-on-the-eye page layout suitable for the WWW, and (2) to make it very easy (for educators) to create interactive exercises and quizzes in the PDF format. Both packages are available in `webeq.html`.

The `web` package (for LaTeX) is a set of macros that establishes a page layout for a (PDF) document that is meant to be read on-screen and *not* meant to be printed. The package also redefines the table of contents to a web style and defines optional navigational aids. The package has options for use with dvipsone, dvips, and PDFTEX.

The `exerquiz` package defines three new environments:

- the `exercise` environment creates on-line exercises; solutions are hyperlinked to the question;
- the `shortquiz` environment is used to construct multiple-choice quizzes (with or without solutions) with immediate feedback in the form of "Right" and "Wrong" message boxes appearing on-screen;
- the `quiz` environment is used to write longer multiple-choice quizzes that are graded by JavaScript.

The `exerquiz` package is somewhat independent of the `web` package; it can be used with any of the standard LaTeX classes and works correctly, for example, with `pdfscreen`,[8] a screen design package written for PDFTEX by C.V. Radhakrishnan.

It should be remarked that both packages have been extensively tested using the commercial TEX system by Y&Y, which uses dvipsone, and the MikTEX system for Win95/NT, which includes dvips and PDFTEX.

Final remarks

It is surprising what can be done with TEX and PDF. In the past, when I thought of TEX, I thought of a compiler and a collection of macros that could produce an outstanding typeset-quality article on paper. TEX, however, is capable of more than just black and white. When teamed with Adobe Acrobat and the Portable Document Format, TEX can produce colorful documents with the richness of user interaction.

TEX and PDF are a natural for putting educational material, especially technical material, on the internet. It is hoped that the use of "team AcroTEX" will continue to grow in the educational and TEX communities.

References

Bienz, Tim, R. Cohn, and J. Meehan. "Portable Document Format Reference Manual". Version 1.2, Adobe Systems, Inc., Mountain View, CA, 1996.

Bienz, Tim and G. Staas. "pdfmark Reference Manual". Technical Note 5150, Adobe Systems, Inc., Mountain View, CA, 1996.

Eijkhout, Victor. *TEX by Topic: A TEXnician's Reference*. Addison-Wesley, Reading, MA, 1992.

Goossens, Michel, F. Mittelbach, and A. Samarin. *The LaTeX Companion*. Addison-Wesley, Reading, MA, second edition, 1994.

Goossens, Michel, S. Rahtz, and F. Mittelbach. *The LaTeX Graphics Companion: Illustrating Documents with TEX and PostScript*. Tools and Techniques for Computer Typesetting. Addison-Wesley, Reading, MA, 1997.

Lesenko, Sergey. "The DVIPDF Program". *TUGboat* **17**(3), 252–254, 1996.

Merz, Thomas. *Web Publishing with Acrobat/PDF*. Springer-Verlag Berlin, 1998. Chapter 6, "pdfmark Primer", is available on-line from: `www.ifconnection.de/~tm/pdfmark/`.

Salomon, David. *The Advanced TEXbook*. Springer-Verlag, Berlin, 1995.

Sojka, Petr, H. T. Thanh, and J. Zlatuška. "The joy of TEX2PDF — Acrobatics with an alternative to DVI format". *TUGboat* **17**(3), 244–251, 1996.

Story, D.P. "Pdfmarks: Links and Forms". On-line documentation: `lnk_forms.html`, 1998a.

Story, D.P. "Using LaTeX to Create Quality PDF Documents for the WWW". On-line documentation: `latx2pdf.html`, 1998b.

[8] CTAN: `macros/latex/contrib/supported/pdfscreen`.

Doing It My Way: A Lone TEXer in the Real World

Jean-luc Doumont
JL Consulting
Achterenberg 2/10
B–3070 Kortenberg, Belgium
JL@JLConsulting.be

Abstract

While a world-renowned standard in many academic fields, Don Knuth's much acclaimed typesetting system is almost unknown in most parts of the real world, where many a document designer has achieved professional success without ever hearing (let alone pronouncing) the word "TEX". Outside academia, the lone TEXer faces not only compatibility headaches, but also outright incomprehension from his customers, colleagues, or competitors: why would anyone want to use TEX to produce memos, two-color newsletters, full-color brochures, overhead transparencies, and other items — in short, anything but books that contain a lot of mathematics?

As a consultant in professional communication, I have been using TEX for all documents I have produced for my clients and for myself during the last ten years or so. Though it has turned out to be most successful, this approach is seen by most as a mere idiosyncrasy. And yet, the systematic use of my own TEX and PostScript programming gives me three unequalled advantages over using off-the-shelf software: I travel light, I can go anywhere I please, and I guarantee I'll get there.

Introduction

Being someone who (among other activities) produces documents for a living, I often have to discuss software issues with clients, colleagues (or competitors, as the case may be), and service bureaus or print shops. When I say I use "TEX," these people usually first ask me to repeat, then express their surprise. Some immediately dismiss it as "never heard of"; others first ask what exactly it is, before wondering why I would want to use something "that nobody else uses" instead of a WYSIWYG, "professional" application (the one *they* use, no doubt).

My using TEX (and PostScript) for virtually all documents I produce, successful as it may turn out, has puzzled many a TEX user, too. Is TEX really the best tool for documents other than long, structured, or heavily mathematical ones? Is a direct-manipulation, integrated application not better suited to producing short newsletters, graphs, or overhead transparencies? The most qualified answer is likely to be, "Well, it depends."

This paper relates the experience of a long-time lone TEXer in the real world. It explains choices, points out advantages and limitations, and draws lessons from the experience.

Approach and opportunities

The very varied documents my company produces can be grouped into two categories. Some are in-house documents, such as letters, reports, and teaching materials (handouts, lecture notes, overhead transparencies). The others are documents we create on behalf of clients, including newsletters, brochures, corporate reports, and overhead transparencies. Documents of both categories may be black-and-white, two-color, or full-color ones, printed in traditional (offset) or modern (digital) ways, in small or large runs.

To be able to guarantee the quality of the documents we produce for our clients, we have opted to give them non-editable deliverables only — in practice, paper, film, or ready-to-print electronic files (in PostScript, for example). We thus strive to preserve the quality of both the writing and the typesetting against two sources of undesirable alterations: unwise last-minute modifications by the clients themselves, often ruining a consistency they may not readily perceive, and accidental changes caused by an all-too-theoretical "seamless conversion" between platforms or versions of a given software application.

Our insistence on not giving away editable files is a tough one to maintain (admittedly, it has suffered exceptions): clients logically consider as theirs the text they have paid us to write or the format they have paid us to design and produce. Yet our name in the list of credits constitutes our best — if not our only useful — marketing tool: we cannot afford to take chances. We will, of course, ensure that what we create addresses the client's needs while attaining the quality level we strive for.

Though our approach is dictated by quality standards, not ease of production, it does simplify our work — to some extent, that is. Because we need not exchange editable files back and forth with external parties, we are free to choose not only the platform but also the software tools with which we work. More accurately, we must be able to process incoming ASCII files and occasional images in GIF or JPEG format, and to produce PostScript files for laser printers, imagesetters, or others still — all in all, minor constraints. Our software tools are therefore essentially limited to an ASCII editor, an image processor, and, of course, a TEX environment, which we use to produce virtually all our paper documents.

Surprise and frustration

The choice of TEX as all-purpose tool for producing very varied documents surprises those who know TEX and puzzles those who do not. Clearly, it does stem from my previous experience with TEX in an academic setting (Stanford University, which happens to be TEX's cradle), and from my technical education (applied physics) and consequent preference for analytical thinking. Admittedly, it may also partly originate in my pronounced taste for computer programming and, more generally, in my peculiar habit of wanting to (re)do things my own way. Still, the perspective provided by some ten years of successful professional practice vindicates my choice: over the years, I have satisfactorily moved to TEX for more and more types of documents.

Until I entered the "real world," I failed to realize how poorly known TEX is, at least over in Europe. It is known by the more technically minded participants at the training programs I teach in academia and national research centers, though even these people are often unclear about what TEX really is and typically confuse TEX and LATEX. In contrast, it is an unknown entity to clients who entrust us with document-creation projects, even those in research organizations, typically from the Corporate Communication or Public Relations department. These clients, used to mainstream direct-manipulation packages, have difficulties understanding the nature of TEX, the difference between TEX itself and its implementation on a specific platform, and the fact that TEX cannot do much (relatively speaking) until one programs it.

Irrelevant as it may seem, given the lack of compatibility issues in our case, the choice of TEX in an essentially non-TEX world proved a liability to our credibility. I had expected that clients who did not know TEX would be content to judge the tool by the result; I was wrong, for at least two reasons. First, most of our clients are little able to judge the quality of a typeset page; their eyes somehow seem blind to stretched out lines, uneven line spacing, and other basic points. Yet the eventual readers of the documents may well notice the difference (if unconsciously), so producing high-quality pages still matters. Second, our clients judge merit by popularity and hence distrust or even disdain what they do not know; they figure that, if they have not heard of TEX, then it cannot possibly be any good. Consequently, and independently from what they see, they are suspicious of the pages I typeset with TEX. Ironically, they seem infinitely more reassured when I simply tell them I use "a system I programmed myself" rather than "the system developed by Professor Donald E. Knuth".

Distrust of the unknown goes beyond the clients. Many of the print shops with which clients sometimes ask me to work dislike my giving them PostScript files instead of editable ones. PostScript being akin to Greek to them, they somehow feel deprived of their power and will not admit I have done half of the work for them. When anything goes wrong at any stage of the printing process, they are quick to pin the blame on me, insisting to the client that I use a software application nobody else uses, "so it's bound to create problems". (As an extreme case, one renowned print shop even resorted to this excuse after mistakenly using a Pantone color other than the one I had specified in writing.)

People who do understand what TEX is, after some explanations, wonder why on earth I prefer using such an intricate system rather than a much more user-friendly (and better known) package. At first, the advantages of TEX were so intuitively obvious to me I was unable to put them into words. Extolling the line-breaking algorithms, positioning accuracy, or logical markup was pointless: these people either saw no benefit in the features or insisted they could do all of that with their own

text processor. To a point, they were right, of course, even if we all feel that "it's not the same".

A little help from my friends

At a loss for words in the defense of TEX, but convinced from experience that it suited my work so well, I set out to ask other users why they preferred TEX. I contacted the makers of my own TEX implementation, posted questions on `comp.text.tex`, and asked around. The answers were varied but can be grouped into five categories: output quality, flexibility, text-based formats, reliability, and portability.

Not surprisingly, many who answered my post on `comp.text.tex` summed it all up by saying that TEX's output "simply looks better," especially as regards mathematics. Among others, they pointed to such features as transparent use of optical sizes, better hyphenation algorithms, and line-breaking algorithms that act on whole paragraphs, not line by line.

Besides the quality of its output, users love TEX for its flexibility, allowing one to extend its capabilities almost ad infinitum. Some mentioned virtual fonts, custom hyphenation patterns, and non-European languages. Others lauded TEX's superior capabilities for floating inserts and cross-references. Following the theme of the present TEX Users Group conference, some also underlined the parallel writing of printed and HTML documentation.

Less expectedly perhaps, users praised TEX's ASCII roots. A plain-text format, they said, allows easy manipulation with any text editor, easy generation of TEX files by other applications (including preprocessors), and small files (hardly larger than the actual text) that compress well. Some also mentioned the small size of their TEX implementation, compared to that of popular text processors.

TEX's text files, batch operation, and stability over time were seen as the basis of its reliability and portability. In contrast to those of integrated applications, the TEX (source) files can be damaged only by the text editing, not by the formatting, the displaying, or any other downstream operation. Moreover, source files typeset with TEX ten years ago can be typeset again today — on any platform — to produce the exact same output, in the exact same device-independent format. TEX files, being plain ASCII, can also safely be exchanged by electronic mail across the world.

Besides the above observations, the (occasionally emotional) discussion on `comp.text.tex` as a result of my post pointed to two interesting differences. Trivial as these may seem in retrospect, they helped me understand much of the religious wars around software: they are the differences between a software tool's

- claimed and actual capabilities, and
- its potential and actual use.

Reliability and portability are typical issues that differentiate between claimed and actual capabilities. There is no intrinsic reason why an integrated application should be unreliable, although increased complexity and fast-changing versions certainly increase the odds. Similarly, there is no intrinsic reason why the claimed interchangeability between platforms and between versions should turn out untrue. Yet a pragmatic point of view suggests otherwise.

Furthermore, the fact an application offers a given feature does not imply that its users actually use it — often a question of usability. As an extreme example, any graphical application that allows one to position characters precisely anywhere on the page can produce output that matches TEX's — but at what cost? TEX, or so its users say, not only makes some operations even possible at all, it also makes many operations easier than direct-manipulation software.

Actual use is also largely affected by stability. In never-ending debates on software, protagonists who claim their own tool to be more efficient than that of others may well *all* be right, for the tool each masters best is the most efficient for him or her. Yet mastery requires time and users feel little motivated to learn to master a tool that will soon change: fast-evolving software, with pressure or even obligation to upgrade regularly, may offer ever-increasing capabilities, but discourages in-depth learning.

As a final point, the `comp.text.tex` discussion also clarified WYSIWYG (What You See Is What You Get), a concept often confused with that of direct manipulation. If WYSIWYG denotes faithful correspondence between screen view (what you see) and paper output (what you get), then some `.dvi` viewers are the closest to WYSIWYG one can get. Many people, however, actually mean that what they *do* affects the output in a way they can instantly *see*, a characteristic of direct-manipulation software. For accurate positioning (for example, to align two words exactly), direct manipulation may not be the most practical approach.

Agreed, but...

Answers from other TEX users expressed some of my intuitions in words, clarified some of my own mental shortcuts, and triggered new thoughts. I agreed on all advantages presented, but felt something was still missing. Most users enthusiastically put forward that TEX is unequaled for *some* documents; very few suggested they were, like me, using TEX for *all* documents they produce.

Users see TEX as particularly suited to the production of large, structured documents. Its batch process, ability to input files in other files, and virtually unlimited capabilities (such as number of paragraph styles) make large projects easier to handle. The separation of contents and visual appearance that it encourages (via macro programming or use) allows one to focus more easily on content, structure, and style.

To my surprise, many a TEX user I talked to admitted to resorting to direct-manipulation applications for short, less-structured documents (letters being the typical example) because "It's so much easier." I wonder what, specifically, they find easier. Letters — business letters, that is — may be seen, if not as one large, structured document, at least as a uniform collection of documents. Consistency, therefore, is as important to the 250 business letters I write every year as a one-off long report. Such a level of consistency, it seems to me, is more easily and more reliably achieved with TEX than with direct-manipulation software. As an example, the instruction `\JL99001 TME/MDe` typesets a letter header with my company's logo, my name, the date (in the proper language), the letter's reference number (here, `99001`), and the complete address block of my addressee (here, person `MDe` from company `TME`, as specified in a data file).

The major perceived obstacle to using TEX, to which some `comp.text.tex` members rapidly pointed, is its steep learning curve. TEX, or rather TEX packages, are often quite immediate to *use*. Despite a marked aversion for computer programming, my business partner used my TEX macros very readily, even without the user manual I never got around to writing. By contrast, TEX, or rather the fine programming of TEX, is not that immediate to *learn*, as confirmed by Knuth's "dangerous bend" signs in *The TEXbook* (1984). The same business partner was often at a loss when something unexpected occurred: the mental paradigm she had built through practice was insufficient to understand those cases.

Conclusion

Faced with people who equate "most popular" with "best", I stopped saying I work with TEX in casual conversation. Today, if people ask, I usually say I use "my own system" and refrain from explaining further. If people insist, I give them a short document that explains my company's approach, including its choice of software tools (Figure 1). Because they may care little about output quality and because our approach bypasses portability issues, the two pages on TEX and PostScript emphasize the three other categories mentioned above: we say it allows our work to be *fast*, *flexible*, and *reliable*.

While I have been a TEX enthusiast for over ten years, I have never been a TEX evangelist. More importantly, while I am convinced that using TEX my own way is one of my best professional moves, I have never advised anyone to develop his or her own package from scratch, let alone use mine. Some clients, who do notice the superior output TEX allows me to produce, have asked what software tool I was using in order "to buy it for themselves". I have to disappoint them, telling them my "tool" could not really be bought. Other clients, who had heard about TEX, asked me for the documents' source files, so they could convert them automatically to HTML. How could their HTML converter know, for example, that my TEX command `\Cs[137]` produces ^{137}Cs?

Using TEX in the real world, where time and money matter much, may require a dedicated TEX wizard. A well-oiled macro package may save considerable time and money by yielding consistently beautiful documents fast. It may, however, not account for the admittedly limited but nonetheless important special cases. In those cases, real-world users may want to call upon a TEX wizard, sometimes on short notice. How severe a limitation this is depends on management strategy. Learning TEX, in my case, certainly paid off, but it was quite an investment, one that was eased by personal motivation but that may not make sense from a pure business point of view. Still, it has given me an edge in many demanding professional cases, in which I can — actually — do what others can not.

I may remain a lone traveler, but my mind is made up: I will go on traveling light, going anywhere I please, and resting assured I'll get there.

Reference

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

Figure 1 (next two pages) The two pages on our using TEX and PostScript, out of a short document explaining our approach to our clients.

Because we strive for the highest quality, we invested in the best typesetting tools. All the paper documents we produce therefore rely on two highly acclaimed standard codes: TEX and PostScript. TEX is a typesetting system developed by Donald E. Knuth at Stanford University for "the creation of beautiful books—and especially for books that contain a lot of mathematics." PostScript is a standard page-description language developed by Adobe Systems Inc. "for imaging high-quality text and graphics."

Software tools

Taking advantage of their complementarity, we use TEX and PostScript in combination. We use TEX programming and encoding for all aspects of typesetting (arranging type on the page). We use PostScript programming and encoding for elements other than type, such as drawings and graphs. For simple illustrations, we write PostScript code directly. For complex ones, we use PostScript-oriented Adobe products: Illustrator™ for vectorial descriptions such as drawings, Photoshop™ for pixel descriptions such as sampled images.

Software compatibility

Since we go from scratch to final pages, compatibility issues are few, if any. As input from clients, we favor the simplest form of all: unformatted ASCII-encoded text or data (often referred to as "plain text"), readily obtainable with most software today. And as output, we convert the whole document to PostScript code, for printing or imaging on a PostScript device. Our clients, in other words, do not have to know anything about TEX to work with us.

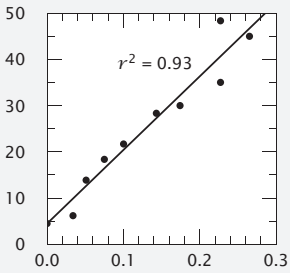
Advantages of own tools

Programming TEX and PostScript—a route on which few professionals venture—gives us three unequaled advantages over using off-the-shelf software: our work is *fast*, *flexible*, and *reliable*. First, we know our tools in and out, and are not slowed down by countless unnecessary options: we travel light. Second, whenever a feature seems to be missing, we program it: we can go anywhere. Third, whenever a feature does not work as we intended, we can and do fix it: we'll get there. We can therefore guarantee a well-done job by a certain date, without fearing that the software irremediably let us down at the last minute.

Freedom has its price: programming TEX and PostScript requires skill and dedication. Skill we have acquired through a high-level technical education, then refined through ten years of practice. Dedication is fueled by our (and our clients') satisfaction with the results. Though we seldom recommend this approach to others, we will continue to develop our own tools, while guaranteeing full compatibility of the output pages with printing devices.

The beauty of TEX

TEX (pronounced "tech" or "teck") is not so much a word processor as a programming language, complete with typed variables, block structure, executable statements, and a powerful macro facility. The TEX compiler turns a one-dimensional source program into a two-dimensional typeset page. Below is an example of source code (right) and resulting output (left).




```
\draw[\white]{8\pc}{8\pc}
\xrange03{10} \xaxis01{10}34
\yrange0{50}1 \yaxis0{10}151
\markerdata
0.034 6.1667  0.051 13.833  ...  \enddata
\linedata
0.0 4.45  0.3 52.28 \enddata
\an{0.2}{35}{$r^2=0.93$}rb
\enddraw
```

TEX's advantages are numerous. Dedicated to high-quality typesetting, it produces the best output available today, especially for such tasks as kerning lines, hyphenating paragraphs, and displaying mathematics. It allows accurate positioning (better than 1 µm) and can tackle virtually any language, in any alphabet. As a clearly defined language, it is platform-independent and stable over time, allows a high level of automation and extension, nicely separates contents from format (thus encouraging logical design, rather than visual one), and works with plain-text source files (smaller, easier to manipulate, edit, transfer, and compress, and much harder to mangle by accident than word-processor files).

The example graph above exemplifies some of TEX's advantages. The plain-text code is compact and compiles fast to produce a consistent graph. Typeface, tick lengths, and line widths are defined at the top of the document, and apply by default to all graphs. The graph is exactly eight interlines high, and is shifted down by $^1/_4$ of an interline; bottom labels are aligned to a line of text, contributing to the overall harmony of the page.

Because typesetting is a complex process, so, unavoidably, is TEX. While merely using existing macros is simple—much simpler, in fact, than using a word processor—writing one's own set of macros is not. Though we would personally settle for nothing less, we consider TEX a tool for professionals and a priori do not recommend its widespread business use around the office.

The vulcan Package: A Repair Patch for LaTeX

Peter Flynn
University College Cork
Computer Centre
pflynn@imbolc.ucc.ie
http://imbolc.ucc.ie/~pflynn/

Abstract

Over many years, TeX and LaTeX systems have proved successful in providing high quality, affordable typesetting from the desktop. This success has tended to disguise some of the less felicitous design decisions made in good faith in earlier days. These are now making it harder to keep LaTeX-based systems in line with user expectations, as the defaults remain rooted in a document model and a rendering which reflect the period of LaTeX's genesis.

Of the thousands of LaTeX packages, many have already tackled the allied problems of providing better formatting and configuration management and of adding new formatting features. This paper presents an attempt to tackle three of the more deep-seated problems: providing an improved document model, an updated rendering, and some fixes for what many users perceive (often wrongly) as bugs.

The document model is compared with current practice elsewhere in the text field, and the fixes answer some of the most common user requests from the various FAQs and questions asked on 'comp.text.tex'. The changes are presented as a standard LaTeX 2_ε package which will be submitted to the CTAN when testing is complete.

Background

When Lamport introduced the LaTeX structured documentation system in 1983, the TeX world entered a new era. There was now a standard set of commands (macros) for processing some common document types, which could be learned very simply, 'even by academics and business types', as an esteemed colleague once phrased it. People who wanted acceptable quality typesetting without having to learn the low-level formatting hitherto used by dedicated typesetters could now obtain it, and LaTeX rapidly became a *de facto* standard for laboratory and other research documents, both in academia and business.

However, as Lamport himself acknowledged during his informal presentation at the Santa Barbara TUG meeting in 1994, the default typographic styling of the article, report, letter, and book document classes leaves something to be desired. The best that can be said about them is perhaps that they formed a standard at a time when no other was available. Regular readers of the 'comp.text.tex' Usenet newsgroup will be familiar with the frequently asked questions about how to change the default styles.

Equally obvious even to the casual user is the lack of some of the most basic features expected by publishers, such as provision for author affiliations and subtitles, and document controls like submission and approval dates.

The concept or model of 'a document' which LaTeX represents is a curious hybrid. Early markup systems tended to regard the document as sequences of text blocks (paragraphs, lists, sections, chapters) separated by headings which applied to all text that followed, until the next heading. The end of a section was thus implicit in the occurrence of a new heading. More recent models, notably SGML, tend to regard the document as composed of textual units which are hierarchically 'containerized'; that is, they have explicitly marked start- and end-points[1] and subsections lie physically within the bounds of their parent section. LaTeX by default uses parts of both models, reflected for example in its uncontained sectioning and in its use of bounded environments.

[1] The quite separate issue of markup minimization for convenience is not relevant: the reference is to fully normalized markup.

Development. For ten years the concept of all but the most trivial changes to the default styles was beyond most LaTeX (and even TeX) users, who had a job to do and were not prepared to commit precious time to making changes in largely undocumented baseline code. The customization features of LaTeX 2.09 were not extensive, and yet it is a tribute to the robustness of Lamport's work and the dedication of hundreds of volunteers that many macro packages were written to extend its functionality.

1993–94 saw the arrival of LaTeX 2ε, intended as an interim solution to some of the problems while work on LaTeX3 continued. The second edition of Lamport's book appeared (1994), updated for LaTeX 2ε, and also the compendious and wonderful *LaTeX Companion*, which provided for the first time a substantial corpus of in-depth detail on the internals of the system, as well as copious information on packages and macros.

Structure. Nothing was done, however, about the level of markup detail or the default appearance of the document styles. This is understandable when LaTeX is viewed from the point of view of a standard, but it is regrettable, as it is the visual appearance of the default styles which is the very aspect of the system which draws the most criticism, and is perhaps second only to broken markup as the reason which has contributed most to the poor image and take-up rate of LaTeX outside the scientific and academic fields.

A second factor in the use of structured documentation systems also came to the fore in the decade before LaTeX 2ε. The SGML standard[2] grew in use (and slowly, too, perhaps for reasons not unlike those which faced LaTeX). The potential for a system which could separate content almost completely from appearance was attractive to many LaTeX users, and those who worked on LaTeX 2ε were very much aware of the existence and growth of SGML. Users, too, were not unaffected by the growing popularity of one rather small and restricted application of SGML called HTML (HyperText Markup Language).

The Vulcanize project. In concluding that there are areas of LaTeX's implementation which need attention, there is no suggestion that LaTeX itself — 'latex.ltx', for example — is in any way 'bust', so there is no implication that the core concepts of LaTeX itself need to be 'fixed'. It is contended, however, that LaTeX's default styles and provision of markup are, at the very least, sub-optimal.

Changing this basic provision of markup is not possible as it is frozen, and too many documents rely on the continued existence of the LaTeX defaults. One solution to this is the development of transparent 'drop-in' styles which provide solutions to the problems without affecting anything else. This allows existing files to be processed without changes to the markup, and allows new files to be created with markup which is 100% backward-compatible.

The Vulcanize project,[3] and the vulcan package which it is developing, grew out of the author's work in supporting LaTeX in a research environment and also using it in a commercial production environment. The simple practicalities of this made it clear that while perhaps 85% of style writing could be handled by existing packages and options, there was a repeated necessity for some common features which were not well addressed by the standard packages. The present macros are therefore aimed at fixing these aspects by supplementing the standard LaTeX defaults with three additional features:

1. extra commands to support many of the requirements for 'logical' markup (Lamport, 1988), not a few of which are derived from the experiences of SGML and XML
2. new formats for the default document classes which become operational without any change to existing markup, drawing on practices and developments in typographic design over the last two decades or more
3. document design support for authors and others wanting to make changes, based more closely on the way the typesetting and publishing industry works

The initial stage of work planned for the project is to cover the first item above, and part of the second. A later stage will tackle the more difficult problems of 'customizable customization' implied by the third.

It has been noted that while the additional provisions of vulcan are in themselves non-standard, there is an opportunity at this stage to try and harmonize some of the work being done by publishers and others, especially for the front matter or metadata, and that the LaTeX3 project may be a suitable forum for discussion.

Other work. There have been many packages which enhance the default TeX and LaTeX styles, and some of them also provide alternative methods of specifying changes to the styles. Among the best known are probably blu, lollipop, and koma. The last in

[2] Standard Generalized Markup Language: ISO 8879:1985 (Goldfarb, 1990).

[3] As was noted in the author's recent column in *TUGboat*(Flynn, 1998), the reference is to the vulcanization of rubber being used to seal leaks.

particular is of interest, as it implemented, for the first time in a TEX system so far as the author is aware, the page layout parameters favored by Jan Tschichold in later life (Tschichold, 1935). These have been borne in mind, although not directly used, in the layout changes implemented in vulcan.

It is no part of this project to detract from these packages, all of which provide much-needed additional features. On the contrary, it is an explicit aim to supplement them by addressing a related but distinct set of problems.

Commands in vulcan

This section deals with the primary stage of the project, which is to develop additional commands to support a greater use of logical markup. The greatest part of this deals with the metadata found in document headers and title blocks. Although the current proposals do not relate directly to those of the Dublin Core[4] or ISO 11179,[5] the author is monitoring developments in this area, and suggestions from catalogers as well as publishers have been sought in relation to the need for markup to hold ISBNs, ISSNs, CIP blocks, etc.

Titling. LATEX's `\maketitle` command creates a title block (or title page, if the `titlepage` option to the current class is in effect), using the values supplied in the immediately preceding `\title`, `\author`, and `\date` commands. If the beginner omits one or more settings, there are defaults or error messages provided by LATEX.

These commands in standard LATEX accommodate only the most basic of document identification, and require sometimes extensive additions to handle the requirements of publishers, institutions, or corporate standards. Code such as that in Figure 1 is all too common, and is largely due to the fact that the only way in the standard classes to effect a subtitle is to bind it typographically to the title, and the only way to effect a multi-author representation is to bind it in a similar manner to the `\author` command by formatting. While it can be argued, correctly, that the default styles can be used to achieve footnoted affiliations, for example, the problem is that they are not called that: they have to be called `\thanks`, which is misleading, especially for the beginner. While it is not possible to provide 'syntactic sugar'[6] for every variation of formatting, the current contention is that the default styles are seriously deficient in their provision of markup and that this is a significant stumbling block to the advancement of LATEX as a serious publishing solution.

One of the greatest problems of acceptance for LATEX is that after all the persuasive sales pitch has been made about the importance of proper markup and identification, and how portable it makes your LATEX, the new user is expected to perpetrate code like that in Figure 1 because the default document classes do not provide for anything else.

Figure 1: Traditional methods of article attribution in LATEX

```
\title{Read Once, Write Many\\
{\Large Disambiguating multi-author
attributions in Restoration comedies}\\}
\author{Mae West$^a$\\
Mae East$^b$\\
Wile E Coyote$^a$\\
\\
{\small $^a$ University of Short
Island, NY}\\
{\small $^b$ Chicken College, RI}}
```

Read Once, Write Many
Disambiguating multi-author attributions in Restoration comedies

Mae West[a]
Mae East[b]
Wile E Coyote[a]

[a] University of Short Island, NY
[b] Chicken College, RI

The problem has long been recognized, and many publishers provide style files for their own journals (*TUGboat* is no exception), but these are clearly very specific solutions, as they address the immediate disambiguation requirement for formatting. It is clear that as publishers slowly move towards more productive use of SGML, the format dependencies are also slowly changing, but in the common case of the author writing a document when a suitable publisher has not yet been identified, many current formats are simply not useful.

For the three basic commands implemented by the standard classes, the vulcan package provides a replacement default title ('Untitled Document') and author ('Anonymous'), and modifies LATEX's date default to use the day–full-month–year format. In addition, it provides for an optional subtitle, and

[4] A proposed methodology for adding metadata to HTML files to aid retrieval on the Web.

[5] A standard proposing Registries to disambiguate the names used for data elements. This would allow context-sensitive searches to take place without the user needing to know what name an information provider has used for a particular class of information.

[6] An alternative way of expressing something; strictly unnecessary, but designed to keep people sweet.

author affiliation details (job title, department, organization, email address, website, and author bio) for multiple authors in such a way as to make the data more congruent with that required by many publishers. This avoids the need for users to add manual formatting to achieve a reasonably acceptable generic article format. As more style files for known publishers' journals or series are developed, it is hoped that a common core of similar markup will emerge: one of the known barriers to acceptance of LaTeX by publishers is the need to make a significant investment in programming in order to realize the required markup.

Author details, if they appear in the title block at all, usually occur in one of two places: a) immediately after each author; or b) grouped together after all the authors, and referenced with a raised footnote letter. To achieve this in vulcan with minimum effort on the part of the author, the positioning of the author details within the markup is used to determine where to place them.

- author details which are included *within* the scope of the `\author` command argument (i.e. before the closing curly brace of the author's name), will be printed 'footnoted' in a block after all author names are complete
- author details which *follow* the closing curly brace of the `\author` command (i.e. they lie after each `\author` command), will be printed as they stand, separately after each author

An example of the first layout is clearly seen in Figure 2. The information which can be provided about each author is defined below (some syntactic sugar has been added to make it more context-sensitive):

`\jobtitle`	Synonyms are `\position` and `\rank`.
`\affiliation`	A synonym is `\institution`, and the abbreviations `\affil` and `\inst` also work.
`\office`	A synonym is `\department`, and the abbreviation `\dept` also works.
`\address`	Postal address.
`\email` and `\webpage`	Email address and Web site, if any.
`\blurb`	This is the author's short-form biography.

The arrangements for handling multiple authors sharing one affiliation is undecided at the time of writing. Different publishers employ very different formats for this, some attaching the affiliation to the first author's name, some attaching it to the last, and some using footnote-style raised figures in multiple occurrences. There needs to be more discussion about how to make it easiest for the writer or editor to assign affiliations to author names.

Markup is also provided for the identification of publication-oriented metadata. This often occurs as a running header, at least on the title page, so it is implemented in vulcan in that position as left-hand and right-hand heads:

`\copyrightholder`, `\copyrightdate`	A copyright notice (holder and year); left head.
`\journal`, `\vol`, `\num`	The name of the journal or other publication, if relevant (with volume number and issue number); left head.
`\conf`, `\sponsor`	The name of the conference or event (and the funder or sponsoring organization), if relevant; right head.

Figure 2: Sample title page in vulcan.

```
\title{Encoding Boole's Algebra}
\subtitle{(Markup and Semantics) or \\
(Markup and Presentation)}
\author{Peter Flynn
        \affil{University College Cork}
        \office{Computer Centre}
        \email{pflynn@ucc.ie}}
\author{Angela Gilham
        \affil{University of Oxford}
        \office{Computer Laboratory}
        \email{angela.gilham@comlab.ox.ac.uk}}
\conf{ALLC/ACH 2002}
\journal{Markup Languages}\vol5\num4
\submitted{31 December 2002}
\maketitle
```

Markup Languages, **5**:4

Encoding Boole's Algebra

(Markup and Semantics) or (Markup and Presentation)

Peter Flynn[a] Angela Gilham[b]

[a]University College Cork Computer Centre (pflynn@ucc.ie)
[b]University of Oxford Computer Laboratory (angela.gilham@comlab.ox.ac.uk)

Submitted: 31 December 2002

Omission of date information works as for standard LaTeX (today's date). Extra commands are provided to identify progress through the editorial process: `\submitted` and `\accepted` are more commonly used by authors, and `\received` and `\reviewed` are more commonly used by editors.

An `\ack` command is supplied to identify funding sponsors or acknowledge other assistance.

In all cases the order of the commands is not important, except that author-specific data must accompany its owner directly, in one of the two places explained above.

Sectioning and structural markup. The majority of LaTeX's default sectioning structure and appearance has been retained, but some aspects which are the cause of numerous FAQs have been changed.

The `sanshead` option sets all section heads, table and figure captions, headings on front- and backmatter sections, and description list topics in the current default sans-serif font. All display section heads are set raggedright at all times.

The concept of a chapter has been removed from code which is applied in the case of the report class, as reports very rarely have chapters.[7] On those occasions when a very large report in chapters is needed, the chaprep option restores the original use of chapters. This not only conforms with observed practice, but solves an important annoyance for the new user.

A new float type, `example`, has been added, using the float package of Anselm Lingnau.

Example 1. Example of an example

The semantics of example markup require some care, as it has been noted that in North America, this term appears to be taken to mean exclusively 'example of verbatim programming code', whereas in Europe it can be any kind of example of anything, depending on the context of the document, from the brushstrokes in a painting by Cézanne to the right way to boil a floury potato.

Formatting

This work forms part of that identified as the second item in the section "The Vulcanize project".

Title block. The layout of the title block or page has been radically altered, as described in the previous section: see Figure 2. Instead of the classic centred design, vulcan uses a flushleft default layout, with sans-serif document title and subtitle, caps and small caps for author names, and smaller type for author details, dates, and acknowledgements.

However, the header layout is just an instance of the `\maketitle` macro, so it can easily be altered to reflect a new design, without the need to borrow heavily from the class file. The remaining work on parameterization (see the section "Conclusions") will address this.

Indentation. LaTeX's automatic `\noindent` after section heads can be extended to new paragraphs following lists, quotations, and so on by using the `trapindent` option. This removes an unpleasant visual imbalance where a new paragraph indent would follow the end of the hanging indentation of a foregoing list. The `flushquote` removes the intrusive initial indentation in the `\quotation` environment.

Extended cross-references. Extended cross-references have been implemented for parts, chapters, sections, tables, figures, examples, (and lists, where possible), so that it is no longer necessary to name the class of object you refer to. If a reference is to a figure, then the word 'Figure' will be inserted automatically. This is similar to the references seen in TeXinfo,[8] which are fully explicit, giving section name and number as well as the page reference, and is already provided for in some class files, as prefixes for counters. The objective is to free authors from the need to carry such details in the head, and to allow text containing labels to be moved from one environment to another without the need to conduct a manual search and check for referential integrity.

The `\ref` and `\label` commands have been modified to handle the relevant additional details, so vulcan files will continue to work with standard LaTeX in this regard, except that the auto-naming will not occur. Work is ongoing to eliminate some minor conflicts with other referencing packages.

In-line lists. One class of lists which is inexplicably missing from LaTeX is the in-line list.[9] These are extremely common in all classes of document, where they a) provide informality, b) consume less space, and c) enable a list to form a part of a complete sentence. They have been implemented in vulcan in the same way as other lists (*ie* as an environment), thus `\begin{inline}`, `\end{inline}`, and `\item` can be used *inside* a paragraph.

There is an optional argument to affect the numbering style, using the same values as for section numbering (`alph`, `Alph`, `roman`, `Roman`, and `arabic`),

[7] It is unclear how chapters ever got into the report document class in the first place.

[8] A plain TeX documentation package common on Unix systems

[9] As this goes to press I notice the new paralist package which also implements these lists.

with a default of italicized lowercase letters (other styles are upright). This argument can also be given in the `\usepackage{inlist}` command if invoked separately, to preset the default.

Hanging indentation. Hanging indentation has been introduced on footnotes by default, using the macro suggested in *The LaTeX Companion* (p. 73).

Conclusions

Some parts of the vulcan package have been in use in the author's work area for many years. Parts of the current version of the vulcan package have therefore already been used to typeset papers and books, and work is continuing on improving robustness and functionality, and on adding compatibility with the darker recesses of the book and report classes.

The objective of codifying these macros was to provide a reliable escape route for LaTeX users who needed a better default presentation format than is provided by the basic installation, and at the same time to explore the possibilities for creating a new suite of document class style implementations to provide new LaTeX users with some visual justification for their placement of faith. It can be an unpleasant experience for someone whose only experience of text processing has been with Word or WordPerfect (or worse) to discover that so much manual labor is needed with LaTeX when its proponents have been expounding its superiority.[10]

Some experiments have been made with full parameterization for all the layout details in the new header, by way of preparation for the next phase of the project. This would allow very simple customization, but the very large number of LaTeX 'lengths' (TeX's `\dimen`*n* registers) needed for all controls would place an unreasonable burden on the use of subsequent packages. The project is open to suggestions about how to achieve a lighter weight solution.

References

Flynn, Peter. "Typographer's Inn." *TUGboat* **19**(1), 353–355, (1998).

Goldfarb, Charles. *The SGML Handbook*. Oxford, Clarendon Press, 1990.

Goossens, Michel, F. Mittelbach, and A. Samarin. *The LaTeX Companion*. Addison-Wesley, Reading, MA, 1994.

Lamport, Leslie. "Document Production: Visual or Logical?" *TUGboat* **9**(1), 8–10 (1988).

Tschichold, Jan. *Designing Books*. Benno Schwabe, Basel, 1935. Quoted in John Lewis, *Typography: Basic Principles*, Studio Books, London, 1963, p. 39.

Sonnet from the ill-at-ease.

Oh what a tangled web we get
when first we practice to typeset.
And so we read TeX master Knuth
Hoping to Obtain some truth.
Or else we read through the book of Spivak
but end up simply crying Alack!
And then we go and **join** the **TUG**
trying to make our process chug.
Beginners know why the rhyme for TeX
is often the frustrated Blecchh.
I can't even get a .dvi file,
and you think TeX to HTML will make me smile?

And yet I know if I abandon the Lion,
With any other package I'll surely be cryin'.

—Joseph Haubrich

The Hidden Passions of Mathematicians

Step into the garden of conjectures and see
my Julia sets are uniformly perfect.
Forget your nilpotence and steenrod algebras,
my theta divisor is very ample.
In this land of lemmata,
you'll glide with the smoothness of Kelley
while I, I'll gather the perverse sheaves
and quivers, and we'll dance
'til our zeta functions converge.
Sipping modular moonshine, we'll reach
the highest eigenvalue without effort.
In this holomorphic vector field
with totally degenerate zeroes,
we may even discover the essence of chaos.

—Debra Kaufman

[10] The words 'mote' and 'beam' seem appropriate here.

New Interfaces for LaTeX Class Design: Parts I and II

Frank Mittelbach
LaTeX3 Project
frank.mittelbach@latex-project.org

David Carlisle
LaTeX3 Project
david.carlisle@latex-project.org

Chris Rowley
LaTeX3 Project and Open University, UK
chris.rowley@latex-project.org

Introduction

Traditional LaTeX class files typically implement one fixed design via ad hoc, and often low-level, (La)TeX code. This style of implementation makes it much harder than is either desirable or necessary to produce classes that implement a specific visual design. Moreover, the construction of such classes typically involves a lot of work that is essentially programming and thus does not live easily with the declarative kind of design specification for a document (or range of documents) that would be produced by a professional typographic designer.[1]

This work introduces some extensions to LaTeX that will help to provide a new, more declarative interface that can be used in class files. It is based on the idea of a *template*, which describes how to carry out some action but which provides some flexibility since its code uses the values of a set of named (keyword) parameters. The specific design for this action, as required for a particular class, is then selected by choosing values for the template's named parameters.

Plans

The plan is to provide standard *templates* for a wide range of typographic objects but, of course, new templates for new ideas can be created, possibly by adapting an existing one or by a little LaTeX programming. It is our firm belief that there will soon be a large range of templates available and that it will thus be possible for the majority of class files to be implemented in a declarative way, by simply choosing suitable templates and supplying values for their named parameters.

Spin-off technology

Whilst applying the idea of templates to document design in LaTeX we have had the opportunity to substantially rethink many of the basic concepts of LaTeX's formatting machinery. This has led to the development of major enhancements in the following parts of LaTeX.

Paragraphs: There will be a completely new model and design interface for all aspects of paragraph-making, including: the parameters that control TeX's hyphenation and justification system; special typographical treatment of the beginning and end of paragraphs, e.g., initial letters/words (lettrines), nested run-on headings, etc.

Galleys: The paragraph model will be linked to a new model for the construction of galleys from paragraphs and other material; this model will incorporate current standard LaTeX concepts such as logical labels, marks and colour-change nodes, together with more experimental objects such as hyper-information nodes.

Floats: These elements have undergone a major redevelopment:

Captions: The formatting and positioning of the caption can be decided individually for each float and can depend on exactly where on the spread it appears.

[1] [This is an up-to-the-minute report from the LaTeX3 team; a written version will follow in a future issue of *TUGboat*. – Ed.]

Position: The position specification allows changes if the float does not fit on the current page.

Pages: More flexibility and better specification of both individual float pages and sequences of float pages (e.g., at chapter ends), including more possibilities to choose whether a text page or float page should be used.

Margins: Better integration of floats and marginal material, allowing floats to appear in margins and for text to be changed according to which margin is used.

Alignment: Better support for alignment between 'minipages', specified using logical handles such as 'top centre', 'centre left' or 'first baseline right': the relative positioning of two boxes (with such handles) is done by choosing a handle on each box and the (2-D) offset between these handles.

Page layout: More types of pages and spreads within a document; more layout choices and parameterisations.

Document commands: New tools for providing document-level syntax.

Professional support: For editorial and page make-up processes currently used in the publishing industry: e.g., flexible manual control over positioning of floats, etc. in the final form document; automated handling of complex bibliographic information in the front-matter of journal articles.

These are all, of course, still severely limited by what is practical within current TeX; for example, precise control over page-breaking within paragraphs is simply unobtainable using TeX's standard mechanisms. Nevertheless, we hope that what we have been able to do will inspire others to use the tools we provide in the creation and use of high-quality typographic designs.

Presentations

The two presentations explain these concepts and show examples of their use, covering both the current standard LaTeX designs and some more exciting new possibilities.

We demonstrate working examples of the application of these ideas in most of the major areas of document design, including page layout, section headings, lists and captions.

We also report on the current status of this work and on our plans to complete and publish it.

Post-Conference Update

The slides from the TUG'99 presentation of the talk we gave on a new interface for LaTeX class designers are available from the LaTeX Project website; look for the file `tug99.pdf` at:

`http://www.latex-project.org/talks/`

The slides are also available from the TUG99 website

`http://www.tug.org/TUG99-web/pdf`

in:

`carlisle.pdf`,
`mittelbach.pdf`, or
`rowley2.pdf`.

(All three have the same contents.)

Please note that the accompanying notes were only intended to be informal "speaker's notes" for our own use. We decided to make them available (the speaker's notes as well as the slides that were presented) because several people requested copies after the talk. However, they are *not* in a polished copy-edited form and are not intended for publication.

Prototype implementations of parts of this interface are now available from:

`http://www.latex-project.org/code/experimental/`

We are continuing to add new material at this location so as to stimulate further discussion of the underlying concepts. As of December 20, 1999 the following parts can be downloaded.[2]

xparse This module contains the prototype implementation of the interface for declaring document command syntax. It supports the definition of user commands with a LaTeX 2_ε interface including star forms, optional arguments, and picture mode arguments. See the `.dtx` files for documentation. It is possible to use this interface independently from all other modules described below.

template This module contains the prototype implementation of the template interface. Together with `xparse` it forms the basis of all further modules, i.e., to make use of any of the other modules you need both.

The file `template.dtx` in that directory has a large section of documentation at the front describing the commands in the interface and

[2] Please remember that this material is intended only for experimentation and comments; thus any aspect of it, e.g., the user interface or the functionality, may change and, in fact, is very likely to change. For this reason it is explicitly forbidden to place this material on CD-ROM distributions or public servers.

giving a 'worked example' to build up some templates for caption formatting.

xcontents This module contains the interface description for table of contents data, describing both an extended data model to replace the data deposited by heading commands into a `.toc` file and a number of template type descriptions for manipulating such data. There is currently no code provided to produce specially formatted table of contents; however, usable examples for the templates have been thoroughly discussed on the `latex-l` list, which can be retrieved as explained below.

xfootnote This module contains documented working examples for generating footnotes, etc. It implements some of the functionality of the package `footmisc` by Robin Fairbains and provides, although still incomplete, a good introduction to the usefulness of templates in class design. Due to the fact that support modules for paragraph manipulation, etc. are still under development, most of the implementation should be considered as a first draft only.

Modules currently under development include the following. Some if not all might be publicly available by the time you read this issue of *TUGboat*.

galley2 This module will document a new data structure for galley-related information, i.e., a data structure to better support handling of inter-paragraph material. Beside implementing lower-level programmer mutator functions for this data structure it will provide higher-level templates for declaring paragraph shapes and H&J (hyphenation & justification) specs. Most other modules will depend on its services as paragraph handling is needed for most aspects of layout. The code of the second prototype implementation for the galley data structure is 90% finished and it is targeted for a release date in 1999.

xlists This module documents and implements templates for providing various kinds of list structures. As part of the included examples it will contain a full set of LaTeX 2_ε lists compatible in design to the `article` class implemented as instances of the templates provided. The prototype code for this module is finished but requires services from `galley2`.

xinitials This module implements a template for paragraph initials. An example of its use can be admired in the slides of our talk. Since it too relies quite heavily on `galley2` it is not yet released.

All examples are organised in subdirectories and additionally available as `gzip tar` files.

These concepts, as well as their implementation, are under discussion on the list `LATEX-L`. You can join this list, which is intended solely for discussing ideas and concepts for future versions of LaTeX, by sending mail to

`listserv@URZ.UNI-HEIDELBERG.DE`

containing the line

`SUBSCRIBE LATEX-L` *Your Name*

This list is archived and, after subscription, you can retrieve older posts to it by sending mail to the above address, containing a command such as:

`GET LATEX-L LOGyymm`

where `yy`=Year and `mm`=Month, e.g.,

`GET LATEX-L LOG9910`

for all messages sent in October 1999.[3]

[3] No, we don't know whether or not the listserv software is Y2K-compliant.

Multi-Use Documents: The Role of the Publisher

Kaveh Bazargan
Focal Image Ltd.
2 St John's Place
St John's Square
London EC1M 4NP
kaveh@focal.demon.co.uk

Abstract

Publishers now expect a variety of electronic files from the typesetter, both for publishing, and for archiving. And yet, the publishing industry, by and large, still follows the traditional "manuscript/edit/typeset/proofread/print" approach. The process is still essentially paper-based. The typesetting is also geared primarily towards paper output.

I suggest that we are in need of radical changes to these procedures so that files can be used to produce output on any medium, including paper, and I believe that these changes will benefit all concerned — authors, publishers, and typesetters. And clearly TEX is an ideal medium to hold the definitive text in modern typesetting. Not only can it be directly edited by the author, but it can produce every type of output required directly. These include paper, PDF, SGML, HTML, XML, etc.

Recent changes in typesetting author manuscripts

Some ten years ago, the process of typesetting material for publishers did not, in general, involve any form of electronic files, whether for typesetting or archiving. The author submitted a paper manuscript which was copy edited and sent to the typesetter for keyboarding, conventional proofreading, and setting to bromide. The procedure had evolved over decades, and if each party performed their roles and knew their responsibilities, the process worked well.

During recent years, the well-defined roles of the three parties — the author, the publisher, and the typesetter — have been blurred, due to electronic files, including those *submitted* to the publisher from the author and those *requested* by the publisher from the typesetter. When we look at current procedures employed by publishers, we see that in the main, the traditional manuscript-based approach is still taken, with the electronic files being treated as an afterthought, once the paper camera-ready copy (CRC) has been completed.

I would like to examine the use of electronic files, and propose some changes in the traditional procedure which should benefit all three parties.

Why use electronic files from the author? For the purposes of this discussion, I shall confine myself to TEX and LATEX files submitted by the author, although the principle applies to other file types.

There are two reasons why files submitted by authors should be used in typesetting a manuscript:

- By using the author's original code, typographic errors can be minimized. TEX documents are often complex mathematical or technical ones, and proofreading them is a difficult task. It therefore makes sense to use the author input where possible.
- If the author has used LATEX in a structured manner, then there may be a significant labour and therefore cost saving for the publisher.

Electronic files required from the typesetter. When publishers first requested electronic files from typesetters, it was limited to PostScript files. These were used firstly in order that the printer could produce high-resolution CRC to print from and, secondly, for archival purposes. Soon afterwards, PDF files were requested, having the advantage of smaller file size and screen viewability. After discovering the joys of hypertext links in PDF files, some publishers requested that these be included for citations, figures, tables, and sometimes to external URLs. Next came HTML and SGML, either for the full text or for abstracts and/or references. No doubt next in line will be XML, MathML, and who knows what will come after that. Gradually, these requests have increased the workload of the typesetter, often without any increase in prices charged.

Using LaTeX to produce electronic files. Fortunately, we find that we can use LaTeX itself, in conjuction with many programs which are in the public domain, to generate all the electronic files required. It is the open code of TeX and that of the auxiliary programs written by third parties that allow us to use LaTeX in this way. The electronic files that can be produced in this way include full SGML files. An important advantage is that there is only one source code and the chances of differences between different electronic versions are minimized. Most of the tools needed for producing the electronic files are, in fact, in the public domain, with the source codes available. This means that they can be customized to produce specific electronic output, for example SGML output for a particular DTD.

The publisher's role

Here are what I think the publishers should be doing to help deal with electronic files better.

Take a more active role in encouraging good LaTeX submission. In order that files prepared by the author can be used easily for multiple outputs, it is essential that these files are coded with document structure in mind and the coding is completely logical, with little or no visual encoding. LaTeX, used correctly, is a very good authoring environment for this type of coding.

It is my feeling that the acceptance of TeX files by publishers has been under the pressure of authors, rather than being initiated by the publisher. With a few notable exceptions, publishers have simply passed on any TeX or LaTeX files received from authors onto typesetters, in case they can be used. By supplying the author with an "author kit" (see below) authors will gain confidence in the publisher and will be encouraged to submit LaTeX manuscripts according to the publisher's requirements.

Recognize the extra care needed when multiple outputs are required. More and more publications are available electronically, usually through the Internet. These include HTML, and PDF files. I believe that publishers should consider all forms of electronic delivery at the outset, and treat the printed CRC as just one of those possible outputs. They should take into consideration the extra care that should be taken in the preparation of manuscripts and their associated files. This means that they should be in contact with the author from the early stages, making sure that clean, structured files are submitted. If such files are not available, then the publisher should be prepared to compensate the typesetter for the extra work incurred.

They could also insist that all outputs, whether electronic or paper, should emanate from the same source code. This will guarantee uniformity in content of the different outputs and minimize delays for last-minute changes.

Distribute an author kit By being provided with such tools, the author is encouraged to submit a well-structured LaTeX document. Here is what the author kit could contain:

- the LaTeX manual [1]: this is a small investment which I believe will encourage the user to follow the standard) — it would only apply to book authors
- an authors' guide, based on the same user friendly approach as Lamport's manual
- a generic class file for authors only (see below)
- a fully working example file with accompanying hard copy

Produce an 'Editors' Guide to Electronic Submissions' Book manuscripts are normally sent to freelance copy editors before typesetting. The copy editors are generally unaware that there are files accompanying the text, let alone that the files may be TeX or LaTeX. It would be useful to produce a short non-technical guide for copy editors, in order to make the process of copy editing smoother and to reduce the number of marks made on paper. Points that should be addressed in this guide would be the following:

- *Automatic cross referencing*: It is useful for a copy editor to know if cross references for citations, equations, etc. are generated automatically. If an equation is deleted, for example, they can simply ask for the rest to be renumbered appropriately, rather than marking each occurance, which is the usual practice. They should also be warned of the reasons for mysterious double question marks appearing in a manuscript.
- *Table of contents and Index*: In LaTeX documents these are usually automatically generated. Therefore copy editing them is invariably a waste of time both for the editor and for the computer operator. In particular, it is very common for page numbers in index entries to be edited. For the typesetter, this is extremely laborious work to carry out and to check.
- *Running heads*: These are also automatically generated and should not generally be marked by the copy editor. By understanding the overall mechanism of running heads, a few simple instructions should suffice for any changes.

- *Making global changes*: The copy editor should be encouraged to make as many global marks as possible, rather than marking every occurance of an error.

The class/style files

The current situation. Let us take the case of book production. At present, a typical scenario is that the publisher asks a TEX consultant to write a class file (let us assume it is only LATEX 2ε we are dealing with). The consultant will be supplied with an example book and/or type specifications, from which to produce a LATEX class file. Normally instructions and an example file are also included. This collection of materials is then distributed to prospective authors to apply to their manuscripts. When the manuscript is received by the publisher, it is sent out for copy editing, and usually goes to a TEX typesetter for production of final CRC and any subsequent electronic files.

The important thing here is that there is only one class file. It is used by the author to produce the manuscript and by the typesetter to correct and paginate the book. I would like to argue that this approach should be re-examined and that separating the author's and typesetter's class files might be a better route to take. Let us look at the requirements of each party in turn.

Requirements of the author. The author's task is primarily to concentrate on the contents of the book or article being written, without much regard for the final look, which is normally decided by the publisher and is the responsibility of the typesetter. Here are what I see as the main attributes of the class file distributed to the author:

- *Standard input syntax*: The LATEX `book` class is well known, and easy to use. It is a great advantage to present an author with a class file which has an input syntax as close to `book.cls` as possible. In fact, the class file and any instruction material should encourage the author to enter standard LATEX code.
- *Easy installation*: TEX is available on most computer systems, including some old machines with limited memory and power. It is a good idea to have a class file that does not need a lot of memory, computer power, or unusual fonts. In particular, it is safest to limit the font requirements to the standard Computer Modern fonts which are available on every TEX installation.
- *Avoid unusual elements*: In order to make it easy for the class file to be used with any system, unusual elements such as graphics should be avoided and be reserved for the typesetting stage.
- *Forgiving class file*: So that the author can concentrate on the contents of the work, the standard TEX parameters for such items as line and page breaking penalties should be relaxed so that overfull boxes are kept to a minimum.

Requirements of the typesetter. Let us now look at the requirements of a class file used by the typesetter. In general, the typesetter should have a more in-depth knowledge of LATEX than the author. He/she can therefore deal with much more complex class files, with the following possible attributes:[1]

- *Unusual fonts*: The TEX typesetter will have access to numerous commercial fonts. The class file can therefore use any standard font for the body text and a choice of several fonts for mathematical text.
- *Graphical embellishments*: I believe that typesetters should strive to get away from the classic "TEX look". There are numerous ways in which TEX and LATEX documents can be embellished with graphical devices, from simple rules to full-colour tints. A TEX typesetter should have all these tools available to improve a book's appearance (in association with a book designer, of course).

Looking at the above requirements for the class files for the author and the typesetter, it is clear that there is a conflict. In the case of the author, the file should avoid non-standard fonts, difficult constructions, graphic elements, etc. On the other hand, the class file used by the typesetter can be as complex as necessary to get the desired effects in the final typesetting. At the moment, any class file writer has to strike a balance between these two conflicting requirements. Graphic elements, if used, are kept simple, so that authors do not have trouble with them. Of course, this limits the complexity of the final typeset book.

Using two separate class files. Looking at the above requirements, it is my conclusion that the author should be supplied with a generic class file, designed with the author requirements in mind. A publisher need not distribute more than one or two of these generic files, making support, debugging, and maintenance easier. The class file used in the

[1] Of course this may not apply to all cases. Some authors are very adept at handling class files, and some books have such an unusual layout that the final class file must be used at all times.

final setting of the book need not concern the publisher at all but should be the responsibility of the typesetter.

Re-use of typeset files by the author

A unique advantage of using LaTeX to set books is that once the typesetting has finished, the files can be passed back to the author to work on another edition. This cannot be done when other systems are used for typesetting, as no other typesetting system can generate clean, structured LaTeX.

Sequence of events in setting a book with LaTeX. Here is what I see as a possible sequence of events, regarding the LaTeX files used in setting a book.

1. Author signs contract with publisher.
2. Publisher sends author the LaTeX author kit.
   The author is encouraged to contact publisher for assistance at this or any subsequent point; typesetter could be called on to support the author in the interests of extracting clean files from the author.
3. Author sends sample chapter to publisher, typeset using generic class file.
4. Typesetter evaluates and reports on sample, with suggestions to author.
5. Author submits manuscript, accompanied by full set of files.
   Note: A possible extra stage here is that the typesetter reformats the files in the final style before handing over for editing, rather than the generic style file being used, exactly as submitted by the author; there are pros and cons to this stage.
6. Copy editor edits manuscript, having read the publisher's 'Editors' Guide to Electronic Submissions'.
7. Author reviews editing (optional, for books only).
8. Manuscript and files go to typesetter.
9. Typesetter implements editor's corrections, converts to final style, and sends copies to publisher.
10. Typesetter receives corrected proofs for CRC, produces CRC and any specific electronic files, and sends these to publisher and printer.
11. Typesetter strips out pagination codes used in the author files to make the final pages, and returns these files to author (via publisher).
12. Author uses generic style file to carry on working on other related material, such as the next edition of the book.
13. Go to step 5 to repeat cycle.

References

[1] Lamport, Leslie. *LaTeX: A document preparation system*, 2nd ed. Addison Wesley Longman, 1994.

The Journey of the TeXxies

"A cold coding we had of it,
Just the worst package from TeXLive
For a document, and such a long document:
The macros long and the braces many,
The very pits of TeX."
And the tables hard, badly-aligned, refractory,
Misplaced `omit` in every `\multicolumn`.
There were times we regretted
The dependence on longtable, the use of pdfmark,
And the hyperref macros breaking our `\cite` s.
Then the authors cursing and grumbling
And using Macintoshes, and wanting their
Framemaker and Word,
And the editor crashing, and the graphics corrupted,
And the `setup.exe` hostile and CTAN unfriendly
And the usergroups dirty and charging high prices:
A hard time we had of it.
At the end we preferred to work in XML,
Validating as we went,
With the voices singing in our ears, saying
That this was all folly.
Then at `</tei.2>` we came to the end
of the process,
valid, with all elements ended, all IDREFs satisfied,
With an XSL stylesheet, and even a version for IE5
And three HTML versions already on the Web.
And an old white horse galloped away in the meadow.

—stolen by Sebastian Rahtz

(untitled)

I thought I would never see
poetry, written by Chimpanzee
But given TeX and infinite time
I'll steal it and call it mine

—Donald Arseneau

Very Like a Nail: Typesetting SGML with TEX*

Frederick H. Bartlett
Springer-Verlag New York, Inc.
fredb@springer-ny.com

Abstract

At Springer-Verlag, we have been frustrated for some years now with the difficulty of putting mathematics into a web-friendly format. We have not yet found a magic bullet, but ...

The XML application MathML may be the first real tool for putting mathematics on the Web in a useful form. Suppliers of mathematical tools such as Mathematica and Maple are gearing up to use MathML as an input/output format; thus, we can look forward to a day when mathematics on the Web will be truly interactive.

It is likely that — even if MathML fulfills every bit of its promise — TEX will continue to be used for the preparation of mathematics for display and printing.

This presentation is an account of our efforts to translate author-generated LATEX into XML. The project can be divided into four stages:

1. Normalizing (La)TEX. That is, transforming authors' idiosyncratic usages (and even more idiosyncratic macro definitions) into consistent, and consistently structured, files. The vast majority of author-generated LATEX files can be converted easily with a minimal understanding of TeX's digestive tract; those which can't (especially plain TEX files) will require some human intervention — or increasingly sophisticated (read 'bloated') software.
2. Converting to XML. This is the easy part: changing structural LATEX tags into XML tags.
3. Converting to MathML. And this is the hard part: It would be ideal to be able to convert LATEX math into both presentation and content MathML coding. Unfortunately, this is, even in principle, extremely difficult. So at first we concentrate on the LATEX-to-presentation mark-up path. Eventually, it will be possible to produce an interactive LATEX-to-content mark-up converter for authors.
4. Going backwards. It will eventually be helpful to authors and publishers if MathML/XML can be converted back to (La)TEX, but this is not a high priority at the moment.

This talk describes something that is very much a work in progress, so a discussion period will be most welcome.

To TEX or not to TEX

To TEX, or not to TEX: that is the question:
Whether 'tis nobler on the page to suffer
The slings and arrows of outrageous software,
Or to write code against a sea of troubles,
And by opposing end them? Use Word? Use Quark?
No more! For such as they could never end
The heartache and the thousand unnatural shocks
That type is heir to.
LATEXe?*
Devoutly to be wish'd! Or Quark to TEX?
To TEX? Now there's a dream. And here's the rub:
From that disguisèd TEX the dream may come
That TEX should shuffle off this mortal coil,
So should we pause? There's no respect
For TEX in all of its long life;
For who would bear the whips and scorns of Frame,
WordPerfect's wrong, Microsoft's contumely,
The pangs of despis'd TEX, Incontext's delay,
The insolence of Active TEX and the spurns
That patient Wizards of th' WYSIWYGers take,
When he himself might his quietus make
In a plain TEX style? Who would authors bear,
To grunt and sweat under a weary life,
But that the dread of something after TEX,
The undiscover'd standard from ISO
And W3C, puzzles the will
And makes us rather love the type we have
Than fly to others that we know not of?
TEX's enterprise of great pith and moment
With XML its currents join anon,
And gain the funds of moguls.

—stolen by Fred Bartlett

* [No paper submitted. – Ed.]

Making a Book from Contributed Papers: Print and Web Versions

Harry Payne
Space Telescope Science Institute
3700 San Martin Drive
Baltimore, MD 21218, USA
payne@stsci.edu
http://www.stsci.edu/~payne/

Abstract

This paper describes a set of tools used for several conference proceedings projects. Processing of a run-file for the individual components is managed by the UNIX make utility, and performed with perl and sed scripts. One advantage to this scheme is that references to other papers in the same volume may be supplied with a page number in a straightforward way. Another is that the entire book may then be processed with `latex2html` to produce a Web version from the same set of input files. The tools will be made freely available via the Internet.

Introduction

In my field (astronomy), it is common practice for conference proceedings to be generated from a set of contributed papers, each of which is a stand-alone LaTeX document. Editors are expected to not only edit the individual contributions but also combine them into a book, with a table of contents and proper page numbers. But publishers generally provide little more than a style file and instructions for the individual authors.

In this paper I will discuss my experience with turning LaTeX manuscripts into a book and into an on-line volume on the Web. I will describe some tools I have written for this purpose, after giving some general comments that may save you some work if you are faced with a similar task. But first, I will describe how I got started on this, and the choices and constraints we faced.

My job is in astronomical software, and in 1994 my institute hosted the fourth annual meeting in a series devoted to Astronomical Data Analysis Software and Systems (ADASS). I volunteered to be a proceedings editor, at least in part thinking that I could enjoy the meeting since my work would all come later (unlike TUG, we go to the meeting first, and then write our papers). The proceedings of the first three meetings were already published, so we knew what our book had to look like. The Astronomical Society of the Pacific (ASP) provided instructions to the authors. Previous editors used perl scripts to kept track of the number of pages in each contribution, so that each paper could be printed with the right page numbers, and a table of contents and index could be generated. But I chose to depend on LaTeX itself as much as possible.

The ASP kindly allowed the editors of the 1993 volume to translate the proceedings into HTML for access over the Web. The Web was new in 1993, and although this first Web volume was attractive, I wanted to do things differently. The first version of Nikos Drakos' LaTeX2HTML had appeared in the meantime. I decided that I wanted to do the entire book as a single LaTeX document and then feed the whole thing to LaTeX2HTML.

I settled on a scheme to edit each contribution as a stand-alone LaTeX document, but to add information for use in the book, such as index and author index entries, and cross references between contributions (fairly common since the authors have already heard one another's talks). The contributions are preprocessed into chapters, and a skeleton document pulls in each chapter to make a book. Front and back matter are created by LaTeX in a straightforward way. The UNIX make utility keeps track of all the pieces, updating the book if any of the papers is changed.

The on-line version of the book is produced from the same files used to produce the printed volume, processed by LaTeX2HTML. The output from LaTeX2HTML is modified with the help of some perl scripts to obtain the final product. Reprints of the papers and a searchable index are provided.

I have used this scheme on about a half dozen books, and other editors have used it on a few more. The editors of the most recent volume in the ADASS series[1] have made some improvements that I will mention.

[1] D. Mehringer, R. Plante, and D. Roberts, University of Illinois.

Editing the Book

As I was editing my first book with this scheme, I made notes to myself every time I went through all of the contributions to fix something. Here are some examples:

- "Took out multiply named labels."
  The instructions to authors said to label your first figure with `\label{fig-1}`. When I put all of the contributions together into one document, `fig-1` was used many times, of course.
- "Added author index information."
  I cloned the `\index` macros to provide a way to compile an author index.
- "i.e. and e.g. are followed by a comma, and are not italicized." "Em dashes don't have spaces around them." "No bold, italics, or Greek symbols in titles or section headings."
  And many other substitutions as well.
- "Work on tables."
  The editors decided that each table would have two `\hline`s, the headers, one `\hline`, the body, and two final `\hline`s.
- "Work on references."
  Describing the new *TUGboat* macros, Robin Fairbairns writes "Bibliographic citations give much grief to the editorial team" (1996, p. 286). This is a mild statement of the situation faced by editors in technical fields. North American and European astronomers use different citation formats, and BibTeX is rarely used.

The common thread connecting these items is that if we had made these decisions and properly instructed our authors before they submitted their papers, we could have saved ourselves a lot of work. Time spent providing your authors with the information they can use will repay itself many times over.

In this spirit, the editors of the most recent volume in the ADASS series came up with a way to add cross references to other papers in the same volume. Authors were instructed to label their papers with the identifier given in the meeting program, and to use this identifier to refer to other papers. In the absence of such instructions, inserting these cross references requires searching for all variations of "this meeting," "this conference," "these proceedings," "this volume," etc. These editors also required authors to make their own author index entries. Instructions for all recent ADASS volumes describe how to refer to Web resources in ways that turn into live links in the on-line version.

You cannot give authors enough information to make their own entries in a real index — making an index is an iterative process that depends on seeing all of the entries. We provide a `\keyword` macro for authors to provide a half dozen keywords or so, just as a starting point for the editors. Our astronomical software series now has enough volumes that we can ask authors to use previous volumes as a guide, but authors must be free to create.

Once the papers and their associated figures have been collected, you can finish editing them only after deciding the trade-off between uniformity and the author's own voice. Making all of the tables look the same was a worthwhile effort. On the other hand, we foolishly once replaced all British spellings with their American equivalents. In the case of authors who are not native speakers of English, you will have to decide when your voice is preferable to the author's — when unidiomatic phrasing is more confusing than colorful.

Finally, if PostScript figures accompany the papers, plan on spending time fiddling with the figures to make sure that the individual papers all print on your system.

Making the Book

Once you have finished editing the papers, you can create the structure for making the book. You need to convert each paper from a stand-alone document into a chapter in the book and create a skeleton document for pulling all of the pieces together.

Depending on the tools you have available, you may not have to wait until you have final versions of the papers. I work with the UNIX operating system, and the scheme I use depends on the make utility program. Programmers use make to manage software projects. You tell it which components depend on which others and specify rules for building the dependent pieces from the independent ones. A program might depend on a number of files containing source code. After editing any of the source code files, the programmer runs make to recompile only the changed files and then build an updated version of the program.

Each paper's `.tex` file is preprocessed to become a chapter in a file with the same name but using the `.ltx` extension. In the make configuration file (Makefile), I list all of the `.ltx` files I will need

```
PAPERS =  \
   accomazzia.ltx agafonovm.ltx \
   alexova.ltx antunesa.ltx \
   ballesterp.ltx ...
```

and provide a rule that accomplishes the preprocessing. Below is an example using sed, the UNIX stream

editor, just because all of the pieces are visible; you could use perl or a compiled program instead.

```
.tex.ltx:
   sed "s/\\\\documentstyle/{% &/" $< |\
   sed "s/\\\\begin{document}/% &/" |\
   sed "s/\\\\nofiles/% &/" |\
   sed "s/\\\\title/\\\\chapter/" |\
   sed "s/\\\\begin{abs/\\\\label{$*}&/"|\
   sed "s/\\\\end{document}/}% &/" > $@
```

To translate, this is a rule for converting a `.tex` file into a `.ltx` file. Each line performs a substitution, commenting out `\documentstyle`, `\nofiles`, and `\begin` and `\end{document}` commands, replacing `\title` with `\chapter`, inserting a label containing the name of the file ahead of the abstract, and putting braces around the entire paper to limit the damage when authors define their own macros. "`$<`," "`$@`," and "`$*`" are `make` macros for the source file, target file, and filename root, respectively.

For the skeleton document, I found it convenient to define a `\paper` macro which, in addition to an `\input`, gives the list of author names as they should appear in the table of contents, as well as the title and list of authors as they should appear in the headers:

```
\paper{accomazzia}{A. Accomazzi, G.
   Eichhorn, M. J.  Kurtz, C. S. Grant
   and  S. S. Murray}{Accomazzi, Eichhorn,
   Kurtz, Grant and Murray}{Mirroring the
   ADS Bibliographic Databases}
```

I like each `\paper` command to be on a single long line, to make it easier to rearrange their order. The ADASS papers alternate author names and affiliations, making it tricky for a program to sort them out. I confess to generally using cut and paste to build these commands, but a perl script can give you a head start. Finish up the book skeleton with a table of contents, and front and back matter of your choice, such as a preface, list of participants, conference photograph, and colophon.

The Makefile says that the book's `.dvi` file depends on the chapter files and the index file

```
book.dvi: $(PAPERS) book.ind
```

As well, the Makefile has rules for obtaining `.dvi` and index files:

```
.tex.dvi:
        $(LATEX) $*

.tex.idx:
        $(LATEX) $*

.idx.ind:
        makeindex $<
```

With these pieces in place, the `make book.dvi` command will create all of the chapter files, run LaTeX to make the `.idx` file, run makeindex to generate the `.ind` file, and then run LaTeX again to make the `.dvi` file. If you go back and edit the papers, you can update the book just by running this command again — only the papers you edited will be processed into chapters again.

A style file for the book defines the `\paper` macro, and redefines the `\chapter` and table of contents macros to match the format required by the publisher. A number of details are also addressed: page format, macros for references to earlier volumes in the series, the author index, and proper numbering of figures, equations, appendices, etc. I wrote macros that add the publisher's copyright notice to the first page of every paper, which I use to produce a version of the book that is sliced up into reprints for the on-line version.

There are Makefile dependencies to control the production of the final printed copy to be sent to the publisher. Our publisher reduces the camera-ready pages we send, so I let LaTeX work with pages and fonts that are the same size as the book, and then ask the dvips program to enlarge the pages to the size requested by the publisher.

In theory, once you have finished editing and produced a final printout, you can send it to the publisher and turn your attention to the on-line version — after all, it is desirable that printed and on-line versions be identical. In practice, I strongly suggest that you wait. At this point, you will be tired of looking at the text, but the first look at the on-line version will make it all fresh again. I guarantee that you will find things to change in the printed version.

Making the On-line Version

Running LaTeX2HTML on a large book does impose some requirements on the hardware you use. Because the program holds the entire book in memory, you will need a large amount of virtual memory — for a 500-page book, I have seen LaTeX2HTML require 750 MB of memory. Aside from this, there is nothing too unusual about the way I use LaTeX2HTML. But I do manipulate its output to produce the final product.

Neither the papers nor the book skeleton file needs to be modified to produce the on-line version. But the Makefile does contain a rule for creating a new skeleton file from the original. The LaTeX2HTML program runs a separate task to expand `\input` instructions; instead of modifying this task I create a

new skeleton file, where all of the `\paper` commands are demoted to ordinary `\input` commands.

LaTeX2HTML allows for customization using perl style files (LaTeX2HTML itself is written in perl). My customizations are either routines to translate new macros into HTML or routines which change the default behavior of LaTeX2HTML. For example, my simple macro for referring to the fifth volume of the ADASS series

```
\def\adassv{in Astronomical Data Analysis
    Software and Systems V, ASP
    Conf.~Ser., Vol.~101, eds.
    G.~H. Jacoby \& J. Barnes
    (San Francisco: ASP)}
```

is implemented with a routine that simply inserts the same text, translated into HTML, into the output stream:

```
sub do_cmd_adassv {
  local($_) = @_;
  join('',"in Astronomical Data Analysis
    Software and Systems V, ASP
    Conf. Ser., Vol. 101, eds.
    G. H. Jacoby &amp; J. Barnes
    (San Francisco, ASP)$_");}
```

The changes to the default behavior are primarily in the navigation panels, where I wanted a particular format, links to an index and reprint files, and a copyright notice. Other changes suppress the index, which I make in a different way, and make sure that chapters are not numbered.

The Makefile has a rule for running LaTeX2HTML on my new skeleton file to produce HTML and image files. Most ADASS papers are short, so I let LaTeX2HTML put each paper into its own HTML file. I also select options which keep footnotes with each paper (rather than breaking them out into separate files) and which add section numbers, an HTML title for the book, and an address at the bottom of every page. Using a Sun SPARCstation 5, running LaTeX2HTML on a 500-page book with 120 PostScript figures and a lot of equations takes the better part of an afternoon. If you have to do it again, it will go much faster if you can re-use the image files. After LaTeX2HTML has done its job, you need to examine the output with a Web browser, looking for problems that might force you to do the conversion again: broken image files or images that are out of sequence.

When you are satisfied with the output from LaTeX2HTML, you are ready to make the final product. I like to rename the output HTML files for the papers, restoring the original name. To do this, I combine information in the `.aux` file from LaTeX and the `labels.pl` file from LaTeX2HTML to create a file containing three columns: the new file name, the original file name, and the page number in the printed version:

```
accomazzia.html node99.html  395
agafonovm.html  node14.html  58
albrechtm.html  node91.html  363
albrechtr.html  node62.html  248
```

I then run a script which renames the files and updates all of the links. The page numbers are used for making the table of contents and index (so you can determine a citation to the book from the on-line version). The table of contents produced by LaTeX2HTML will not contain the author names, so I replace it by running a script on the `.toc` file from LaTeX. While working on the table of contents, I put all of the front and back matter together under a "Preface" heading. I generally make a version of the book cover by hand and use either this cover or the table of contents as the entry page for the on-line book.

One of the advantages of the on-line version of the book is the ability to do full-text searching. There are many options for indexing software, and many are free, but your choice will depend on your Web server platform. I have used WAIS, a somewhat dated search technology, since I already had a WAIS server available. Every page in the on-line version has a link to an index page from which the user can perform a search. Although this mechanism provides a powerful way to find what you are looking for, I also translate the author and subject indices from the book into HTML, using perl scripts and information from the `.ind` file produced by LaTeX.

We published the on-line version of the book with the permission of the publisher. As a courtesy, we put the publisher's copyright notice on each HTML page (making the notice into a link taking users to the publisher's home page) and each reprint. I also added instructions for obtaining a printed copy of the book by way of the publisher's website. Except for the searchable index, all of the links in the on-line version are relative. This makes it easy to create mirror sites.

You can spend as much time as you like on projects like JavaScript commands to identify people in the conference photograph just by moving the mouse. However, you can realistically expect to produce an on-line volume with a few days of effort, once the printed volume is ready.

Trying it Yourself

You can find the files you need to try out this scheme at `ftp://ftp.stsci.edu` in `/pub/software/tex/bookstuff/`.

The most recent on-line volumes produced this way can be found at `http://www.stsci.edu` in `/stsci/meetings/lisa3/` and `/stsci/meetings/adassVII/`.

References

[1] Fairbairns, Robin. "The New (LaTeX 2$_\varepsilon$) *TUGboat* Macros." *TUGboat* 17,3 (1996), pp. 282–288.

TUG '99 musings

Mr. Fine, with the eye of the sleuth
has discovered why TeX is uncouth,
that its catcodes are many
when there shouldn't be any
Has anyone told Donald Knuth?

Mr. Flynn is now running on La-
TeX, and finds, when the tread starts to fray,
he can patch up the rules
with his vulcanize tools
pump them up, and back on his way

Though some of you authors may fret
about how every page should be set
and think what you see
and what it must be
Mr. Bazargan says what you get

—Pierre MacKay

Ode to a special

Oh what a tangled web we get
When first we find `\def` and `\futurelet`.
We turn to eTeX, we turn to Omega,
They both look good, we start to get eager.
But no, whats this, a misplaced `\omit`?
Did we miss a brace, does `\vbox` still not fit?
Help me fix my macros, bracket heroes all!
But Pierre's font is calling, Lamport's left the hall
Barbara says use `\downcase`, Downes just strokes his beard.
Erik sells me 4TeX, pretends he never heard.
David's hacking tables, Frank is fixing floats,
Ogawa's talking slowly, Kacvinsky wants his oats.
Will Kaveh help me out? no, his dhoti's dirty,
Nelson's feeding awks, Mimi's feeling flirty.
Flynn says use a rubber, Wendy needs a hug,
Kath has got the answer, no it's just another mug.
Young Ross is such a hoot, he says use xypic
Why not turn to Sojka, he's sure to know the trick.
Anita, a Hoover, what use to me's a dam?
Send me out with Kiren, we'll both go on the lam.
Irina claims 'for us in Mir is no problem',
TRIUMF uses `\mathcode` but `\hspace`s just one em.
TrueTeX does it both ways, and you can trust the Blue Sky
but hyperref the backend, oh why oh why oh why?

ActiveTeX, PassiveTeX, what a great to do,
Can there be yet some way through?
MathML has pointies, XSL can claim its templates
ExerQuiz is **so** cute—so phooey on you Billy Gates.

—Sebastian Rahtz

Managing Large Projects with PreTEX: A Preprocessor for TEX

Robert L. Kruse
PreTEX, Inc.
2891 Oxford Street
Halifax, Nova Scotia, B3L 2V9
Canada
bob@pretex.com

Abstract

PreTEX is a preprocessor for TEX that supplies an author with many tools to simplify the writing and management of larger (book-length) projects. This paper concentrates on PreTEX's use of secondary input files and conditional typesetting in managing large projects. The user may insert location tags within a file which can then be used by PreTEX to include parts of one file within another, in any order determined by the user. Parts of a file may also be selectively typeset according to the status of various conditions.

Introduction

Let us consider two projects large enough to challenge most TEX systems.

The first is a large science or math textbook at the high school or elementary college level. Such a book is often over 1000 pages long, requires four-color printing, and contains many hundreds of full-color photographs or other complicated images. It has a large index, bibliography, and cross references throughout. Answers to some of the exercises appear at the back of the book. There may be many supplements, such as a separate solutions manual, a study guide, a lab manual, a bank of test questions, or transparency masters. All of these contain many references that need to be keyed to the textbook itself. There may be a separate book of instructor's notes keyed to the text, or the book may be simultaneously published in a special instructor's edition with the notes in the margins or on extra, unnumbered pages. Some of these many supplements may be published on paper, some on the Internet, some in PDF or other format on CD-ROM with interactive links.

Our second example project is the documentation library for a large software system, one that may contain hundreds of thousands or millions of lines of computer code, written in several different programming languages and distributed over many files. To avoid errors and inconsistencies, it is important to keep all the documentation in the same files with the code. Such a system will have several kinds of documentation: informal introductory guides for the user, reference volumes for more sophisticated users, precise specifications for each part of the program, in-house documentation and other comments from programmers, and bug/modification reports and history.

PreTEX is a preprocessor for TEX (consisting of about 15,000 lines of C code), designed explicitly to facilitate the work of authors and editors in writing and production of large projects such as these. PreTEX's features, moreover, remain equally useful for more ordinary book-length projects or even for small typesetting projects such as research papers.

By exploiting context dependency, PreTEX supplies much of the routine markup required for high-quality typesetting in TEX, simplifies the notation for mathematics, supplies user-friendly error diagnostics, uses its own tables of information to resolve many ambiguities in typesetting, and, by recognizing some of the syntax of various programming languages, provides powerful tools for typesetting computer-program listings.

Some of these features were discussed for a preliminary version of the PreTEX software in Kruse (1988). Since that article was published, the PreTEX software has been substantially revised, extended, and used intensively in a commercial environment. It is now ready for initial public release.

This paper discusses only a fraction of the tools provided by PreTEX. Here, we concentrate on PreTEX's file-management facilities that are especially valuable to authors of large projects. For a discussion of PreTEX's implementation of hypertext links in PDF, see Mailhot (1999).

Independent chapter processing

High-resolution scans of color photographs or other complicated graphics require considerable space in computer storage. If a book contains hundreds of such images, its output files can become prohibitively large, often several gigabytes in PostScript. It therefore becomes imperative to divide such a project into smaller files that can be processed independently.

In fact, PreTEX allows a project to be divided into conveniently small, chapter-length files which are processed independently at *every* stage, including the final production of PostScript or PDF files. At the same time, PreTEX integrates the cross references, the index and contents entries, and the bibliographic citations from *all* the input files comprising the entire project. Hence, while the author works on the files for one chapter, cross references to other chapters will still resolve properly. Page numbers, chapter numbers, and other elements that number consecutively throughout the book are automatically updated for each chapter file.

To accomplish these goals, PreTEX maintains a whole directory of auxiliary files in place of the single auxiliary file used by LaTeX. Indeed, for each chapter there are several auxiliary files that are accessed by PreTEX, by TEX, by BibTEX, and by PreTEX's enhanced equivalent of MakeIndex. Under normal circumstances, the user never needs to look at any of these auxiliary files directly. Since there are separate auxiliary files for each chapter, the system modifies only those corresponding to the chapter currently being developed.

For some purposes, PreTEX must string all these files together in the correct order; to do so, PreTEX uses a short file containing a master list of all chapter files. A sample of this master file list is:

```
title
contents
preface
part1
chapter1
chapter2
part2
chapter3
appendix1
appendix2
index
```

The blank line specifies that the page numbering is not continuous from `preface` to `chapter1`; otherwise each file begins after its predecessor. Contents, index, cross-reference, and bibliographic entries, however, are merged for all the files.

Secondary input files

One of PreTEX's most powerful features is its ability, while processing one file, to include portions of other files, arranged in any desired order, with parts skipped under the control of Boolean (that is, true-false) variables. This feature is like TEX's `\input` command, except that `\input` includes the entire file, whereas PreTEX may select only portions.

First, we need some notation.

PreTEX Command Syntax. As a preprocessor, PreTEX has its own extensive command language, as well as responding to certain TEX commands. It recognizes several different environments and processes its input differently according to the environment. In the mathematics, verbatim, and computer-program environments, the characters '`<`' and '`>`' are processed as usual, but in text (where they would normally not appear) '`<`' and '`>`' are used to delineate commands to the PreTEX preprocessor. Such commands take forms such as

```
<:command_name option1 option2>
```

where the number of options and their syntax vary with the command.

To access a secondary input file, we need only write an instruction such as

```
<:read file filename from tag1 to tag2>
```

at the place where we wish to include part of the secondary file `filename`. The location tags, denoted `<tag1:>` and `<tag2:>`, are placed immediately before and after the part of the secondary file that will be read. Any number of location tags may be placed in a file (at any place where the file is in text environment); these tags are used only to control file reading and will not appear in the output after processing by PreTEX.

Conditional Typesetting. When the same material is processed for different purposes by PreTEX, it is often convenient to use TEX macros both to control how an element is typeset and, depending on the purpose, to determine if the element is included at all. In a textbook, for example, the author may wish to place solutions to exercises immediately adjacent to the exercises themselves, so that, if the exercise is modified, its solution can easily be modified to match. When we are processing the textbook itself, these solutions must not be included in the output, but when typesetting the solutions manual, they must appear.

We can accomplish this goal by instructing PreTEX to create a new Boolean variable, which we might call `solutions`, and to use this variable to

control the typesetting of material between a pair of control sequences, which we might call `\solution` and `\endsolution`. With this instruction, PreTEX will recognize two new commands:

`<:solutions on>` and `<:solutions off>`

Depending on the setting, PreTEX will include or delete material between

`\solution` and `\endsolution`

For the textbook itself, we specify

`<:solutions off>`

(likely in a style file, so it need be done only once for the entire book), in which case all solutions will be deleted. When processing the solutions manual, we specify

`<:solutions on>`

so that all solutions will appear immediately after their exercises.

With these tools, typesetting a separate solutions manual is trivial, and it will be guaranteed to be kept current with any revisions to exercises in the text. All we need to do is to place location tags before and after each group of exercises in the text's chapter files. Then the file for the solutions manual will contain almost nothing except for the command `<:solutions on>` followed by a long series of `<:read file ...>` commands to include each group of exercises and their solutions from the various chapter files.

Some authors, on the other hand, prefer to place all exercises and their solutions into separate files, one or more exercise files per chapter. This organization will work just as well. Now, with location tags before and after each group of exercises, `<:read file ...>` commands in the main chapter files will include the exercises where desired (along with their solutions, which will be deleted, provided `solutions` is `off`).

To place answers in the back of the book is just as easy. We now introduce a second Boolean variable, say `answers`, include the commands `<:solutions off>` and `<:answers on>`, and use the same `<:read file ...>` commands to typeset, at the book's end, only the selected answers.

Production of other supplements for a large book follows much the same plan. The material for these supplements can either be placed in the main files with typesetting controlled by Boolean variables, or placed in separate files used to produce the supplements, with `<:read file ...>` commands as appropriate to include material from the main file or other supplements. In any case, the names of the files for all the supplements should normally be included (in separate groups) in the master file list for PreTEX. In this way, cross references may be made from any of the supplements to any other or to the main text. Similarly, in PDF, hypertext links allow the user to jump directly from locations in the text to appropriate locations in any supplement, or the reverse.

Program code and StripTEX

Now let us turn to our second motivating example, a documentation library for a large software system, where the code and documentation are distributed over many different files, generally with differentauthors.

By treating each file of program code and documentation as a secondary file, PreTEX can be used to combine any desired extracts of the documentation or the program code in any desired order. The identical source files can in this way be used to construct, for example, informal user guides, user reference manuals, programmer's reference manuals, or other documentation, either for in-house use or external distribution.

In PreTEX's different environments, the same symbols will be processed in different ways. PreTEX provides special facilities for typesetting computer programs, understanding enough of the program syntax to adjust spacing and choose special symbols. For example, curly braces `{ ... }` (delineating a group in TEX) become printed symbols in C or C++, enclosing code segments, whereas in Pascal, they delineate comments that will be typeset as text, not as computer code. PreTEX provides environments for many common programming languages (Java, C++, C, Pascal, Ada, Fortran, Basic, Cobol, among others). By treating program lines as tab-controlled lines (as illustrated in *The TEXbook*, page 234), tab stops can be used to achieve proper alignment, or other TEX markup can be inserted into programs.

For these program files, PreTEX provides a further utility, called StripTEX, that removes all the text surrounding program code, as well as any TEX markup in the program code, yielding output that can be submitted directly to a compiler. StripTEX recognizes all the same commands as PreTEX, including reading from secondary files. Hence, the order of subprograms within files and their accompanying documentation need not have any connection with the order expected by the compiler; StripTEX can be used to extract subprograms from arbitrary files and arrange them in whatever order is needed by the compiler. The same file(s), containing both documentation and

code, can be processed through PreTEX to obtain a well-formatted, documented program listing, or through StripTEX to obtain the program in exactly the form needed by the compiler.

In this way, PreTEX and StripTEX provide all the functionality and capabilities of Knuth's `WEB` system. PreTEX replaces `WEAVE`, and StripTEX replaces `TANGLE`. The PreTEX–StripTEX system, moreover, brings several advantages over `WEB`:

- The user has unlimited flexibility in the ordering of subprograms and documentation and their placement in various files.
- The same software manages programs in any number of computer languages, including several for which `WEB` is not available. For software systems using several languages, the identical software generates all the files needed for the various compilers.
- The user has no need to learn a new command language: StripTEX shares the identical syntax of PreTEX, which is an easy extension of TEX.

Bibliographic entries

For large projects with many references, maintaining the bibliography can require considerable work. BibTEX provides excellent facilities for maintaining a bibliographic database; PreTEX uses BibTEX without change. Any LaTEX user of BibTEX will immediately be familiar with the PreTEX commands `<:cite ...>`, `<:nocite ...>`, `<:bibliography ...>`, and `<:bib_style ...>`.

As a preprocessor, PreTEX can automate and streamline the use of BibTEX. LaTEX, for example, requires a four-pass process:

1. Run LaTEX to collect the citation keys.
2. Run BibTEX to assemble the corresponding references from the database(s).
3. Run LaTEX to associate the citation keys with their references.
4. Run LaTEX to resolve the citations.

PreTEX accomplishes the same results with only two passes and with no special attention from the user. On its first pass, PreTEX assembles the citation keys and automatically invokes BibTEX, after which PreTEX immediately associates the citations with their references. On its second pass, PreTEX resolves the citations. Whenever the document is processed again, PreTEX automatically detects if the citations have been changed, and PreTEX invokes BibTEX only when necessary to keep the bibliography up to date.

As our final example, consider a symposium or conference proceedings in which the individual chapters are written by different authors who may not be in touch with one another. Each chapter will then be written and processed independently; if desired, each chapter may have its own cross references, index, or contents. The editor may then later merge all these resources for the entire volume. The editor may supply a bibliographic database for use by all authors, accessed as needed by BibTEX from within PreTEX. In addition to these global citations, PreTEX allows bibliographic citations local to each chapter, so each author may use both the global database and, independently, use local citations from other bibliographic databases. To enable local citations, PreTEX includes a second set of citation commands with '`l`' prefixed to the name of each, such as `<:lcite>`, `<:lbibliography>`, and the like.

Preview

This paper touches on only a few of the tools PreTEX provides to authors to simplify the writing, management, and typesetting of large projects. Together with its preprocessor, the full PreTEX software package contains the StripTEX processor, an auxiliary program for the construction of the index, another for the table of contents, and a large TEX macro package of about the same size and capability as LaTEX, but with a somewhat different design philosophy.

The PreTEX software has been under development and revision for several years, and at the same time it has been used intensively by PreTEX, Inc., for the commercial typesetting of many textbooks in mathematics, engineering, and science. The software is now ready for initial external testing and application, with public release to follow in due course. Before its general public release, however, some work remains to be completed, especially the writing of user guides, reference manuals, and other documentation necessary for the effective application of the PreTEX tools.

Bibliography

Kruse, Robert L. "PreTEX: Tools for Typesetting Technical Books." *TEXniques* No. 7: *Conference Proceedings of the Ninth Annual Meeting, Montreal, August 1988*. Ed. Christina Thiele, pp. 219–226. TEX Users Group. 1988.

Mailhot, Paul A. "Implementing Dynamic Cross-Referencing and PDF with PreTEX." 1999. (See elsewhere in these Proceedings.)

Database Publishing with JAVA and TEX*

Arthur Ogawa
TEX Consultants, California
ogawa@teleport.com

Abstract

Sun Computer intends its JAVA programming language to be the lingua franca of the World Wide Web. Sun may get their wish: dozens of books about JAVA are published every year, and there is even a conference about JAVA, JavaOne, being held in San Francisco at the same time as this TUG meeting.

If you wanted to publish a book documenting and cross-referencing all the JAVA class libraries (running to 1,000 pages and covering over 1,500 classes, organized into about 70 "packages") what would you do? If you are Patrick Chan and Rosanna Lee, you would use JAVA itself to manage the data (about 40Mb of it) and you would use TEX to format your pages.

In my talk, I will describe the criteria for selecting the formatter (i.e., TEX plus macros) for a database typesetting project, the best way of interfacing between TEX and a database engine, and some interesting (perhaps even challenging) features of the formatting work. I will also show how the success of the project enabled the author to make last-minute revisions to the book (changes necessitated by late developments in the JAVA class libraries themselves) even though this involved the reprocessing of all 40Mb of data, in less than 24 hours.

Ode to a special

Oh what a tangled web we get
When first we find `\def` *and* `\futurelet`.
We turn to eTEX, we turn to Omega,
They both look good, we start to get eager.
But no, whats this, a misplaced `\omit`*?*
Did we miss a brace, does `\vbox` *still not fit?*
Help me fix my macros, bracket heroes all!
But Pierre's font is calling, Lamport's left the hall
Barbara says use `\downcase`*, Downes just strokes his beard.*
Erik sells me 4TEX, pretends he never heard.
David's hacking tables, Frank is fixing floats,
Ogawa's talking slowly, Kacvinsky wants his oats.
Will Kaveh help me out? no, his dhoti's dirty,
Nelson's feeding awks, Mimi's feeling flirty.
Flynn says use a rubber, Wendy needs a hug,
Kath has got the answer, no it's just another mug.
Young Ross is such a hoot, he says use xypic
Why not turn to Sojka, he's sure to know the trick.
Anita, a Hoover, what use to me's a dam?
Send me out with Kiren, we'll both go on the lam.
Irina claims 'for us in Mir is no problem',
TRIUMF uses `\mathcode` *but* `\hspaces` *just one em.*
TrueTEX does it both ways, and you can trust the Blue Sky
but hyperref the backend, oh why oh why oh why?

ActiveTEX, PassiveTEX, what a great to do,
Can there be yet some way through?
MathML has pointies, XSL can claim its templates
ExerQuiz is ***so*** *cute — so screw you Billy Gates.*

—Sebastian Rahtz

* [No paper submitted. – Ed.]

Implementing Dynamic Cross-Referencing and PDF with PreTEX

Paul A. Mailhot
PreTEX, Inc.
2891 Oxford Street
Halifax, Nova Scotia, B3L 2V9
Canada
paul@pretex.com

Abstract

This paper discusses how we can create dynamic and interactive on-line documents in Portable Document Format (PDF), using TEX. PDF documents are generally device and platform independent, therefore, ideally suited for on-line publishing and information exhange. We will need to work in three programming languages: macros in TEX, and definitions in PostScriptand PDF. We begin by describing the steps necessary to process PDF through TEXand PostScript, followed by several examples of defining such TEX macros. The examples are quite easy to follow; however, some knowledge of PostScript and PDF programming is useful.

Paperless publishing

Publishing of books and documents has dramatically changed in the past few decades. Philip Taylor (1996) accurately described the emergence of computer typesetting with emphasis on Web document rendering applications, using portable multiplatform hypertext exchange standards, particularly focusing on the merits of PDF and HTML (including XML, SGML, et al.). For our purpose we can define *on-line* as any means by which a document can be electronically rendered; this is perhaps more aptly called *paperless printing*.

The popularity of on-line books and documents has spawned a diverse variety of on-line readers. These *readers*, for our purpose, are programs, applets, and interpreters which can *read* electronic data and output the image to a screen, in much the same way that one would see the output from a printer. Readers include proprietary single-platform programs, stand-alone multi-platform readers (including Frame Reader, Ghostscript, and Acrobat Reader), Web browsers (including HTML and JAVA), and Web browser plug-ins (including Acrobat). It is not at all wrong to suggest that TEX, along with a DVI previewer, is one of the first platform independent readers. Many people have generously contributed packages to the TEX cause, by which users can incorporate recent advancements in reader design such as HTML and PDF.

Portable Document Format

Portable Document Format, more commonly called PDF, was developed by Adobe Systems and is a descriptive programming language, devoid of software and hardware dependence, used to render documents. PDF is sometimes grossly misunderstood as being a subset of the PostScript format language. It is true that PDF and PostScript share many common features but each format language suits a particular task, and there are features found in each but not the other. Thomas Merz (1997) describes these similarities and differences in suitable detail. For our purpose we will concentrate on PDF documents, in particular addressing some hypertext features including bookmarks, links, and annotations.

A PDF document in general cannot be directly viewed, but rather, must be processed through a reader such as Adobe Acrobat Reader or Ghostscript. There also exist a number of application plug-ins for internet, word-processing, and desktop-publishing software which allow viewing of PDF documents. PDF documents are usually created by printing a document through a printer driver such as Acrobat PDF Writer or printing to a PostScript file which is in turn passed through Acrobat Distiller. The second method is more commonly used by the TEX community; a brief description is given by Amy Hendrickson (1998). Contributions such as `hyperref` (Thành and Rahtz, 1997) and `pdftex` (Thành, 1998) provide direct-to-PDF document processing.

PDF through PostScript

For our purpose we will concentrate on the method used by Hendrickson. The description of PDF and PostScript operators for hypertext links is found in Merz, but for our purpose we will briefly review it.

In order to create PDF links, we must first supply a set of raw PostScript instructions/definitions which will allow passage of PDF link code through Acrobat Distiller to create PDF files with links. This PDF code will be ignored by non-PDF devices such as PostScript printers. The following PostScript source code accommodates this by an `if–else` statement:

```
/pdfmark where
   {pop}
   {dictionary /pdfmark
    /cleartomark
    load put}
ifelse
```

In this PostScript code, the `where` command searches for a PostScript dict containing a definition of `pdfmark`. If the definition is found, then the `if` portion `{pop}` is executed and word `pdfmark` is popped off the execution stack by the PostScript interpreter; otherwise, `pdfmark` is defined to remove all the tokens between *mark* and the word `pdfmark` by using the PostScript operator `/cleartomark`. A mark in PostScript is the character `[`. We now have an avenue by which PDF code can be passed through both to PDF devices and non-PDF devices.

Most new DVI-to-PostScript interpreters have already included the above PostScript code in their document header. Those interpreters that do not include it can do so with the following TEX definition:

```
\def\initializePDF{
   \special{header=pdfparse.psc}}
```

This TEX macro tells TEX and any generic DVI-to-PS driver to insert the file `pdfparse.psc` into the PostScript output file header. The file `pdfparse.psc` would then contain the above PostScript code with the `/pdfmark` definition.

The PDF code

Now that we have taken care of passing PDF link code, we must create PDF code for creating PDF links. The PDF code works in the same manner as PostScript in that we must create definitions using predefined PDF operators. There are essentially two parts to implementing PDF links. Merz (1997:288–298) gives several examples of fully functional links which can be easily followed. The example

```
[ /Rect [ 0 0 60 60 ]
  /Page 124
  /LNK  pdfmark
```

contains all of the essential elements required for a link to pass through PostScript and be executed in PDF. The PostScript mark `[` and definition `pdfmark` delimit the PDF code. It cannot be assumed that the code between the delimiters is PDF because we may need to pass values from TEX into the PDF code. The fully functioning example

```
\def\pdfbookmark#1#2{{
%#1=section/Chapter/etc..
%#2=usually the title
\special{verbatim=
" [ /Title (#1 #2)
    /OUT pdfmark"}
}}
```

shows how we can define a TEX macro, using a PostScript special, to insert entries into an outline of a PDF document. An example of an outline in Adobe Acrobat is shown in the left side of Figure 1. We can jump to each entry of the document by clicking the icon to the left of the outline entry.

Figure 1: An example of a PDF document containing an outline. We can use the outline to jump to its location in the body of the text.

A more useful but trickier example is:

```
\def\pdfnameddestination#1#2{{
%#1=xrf-tag name
%#2=pagenumber
\special{verbatim=
" [ /Dest   /#1
    /page   #2
    /Border [0 0 0]
    /DEST   pdfmark"}
}}

\def\pdfgotonameddestination#1{{
%#1=xref-tag name
\special{verbatim=
" [ /Rect [ currentpoint
            2 neg add exch
           10 neg add exch
```

```
            currentpoint
            8 add exch
            0 add exch]
  /Border [0 0 0]
  /Dest   /#1
  /LNK    pdfmark "}
}}
```

This example supports dynamic cross-referencing in the PDF document in the same manner as LaTeX. The first macro creates a mark where the source reference is located and the second macro creates a link to that reference. The second macro does this by creating a PDF link box 10 points square in the PDF document. If, in Acrobat Reader, you click on this box, you will jump to the page where the link is defined.

One major restriction of this method is that the entire project needs to be in one PDF document.

Multiple-file PDF documents

Many books and other large projects need to be processed as multiple files, broken up so that each section is of more manageable size. But our earlier method of creating links as named references does not work outside a single file.

A better method, which works over multiple files, is to create a set of macros that reference both the filename and the page number With this method, we can set up a link to any PDF document, as long as we know the file name and the absolute page reference. This method is implemented in the following TeX definition:

```
\def\pdfxrffileopen#1#2{{
%#1= file name
%#2= absolute page number -
% not the printed page number
\special{verbatim=
" [ /Rect [ currentpoint
            2 neg add exch
           10 neg add exch
            currentpoint
            8 add exch
            0 add exch]
    /Border [0 0 0]
    /Action << /Type /Action
      /Subtype /GoToR
      /File (#1.pdf)
      /Dest [ #2 1 neg add /Fit ] >>
    /Subtype /Link
    /ANN pdfmark "}}}
```

This definition contains a few more operators but it has an overall structure similar to the earlier example. It is important to note that PDF documents are referenced by absolute page number. If, for example, the document contains front matter numbered i to xvi and text pages 1 through 23, then the absolute page numbers are 0 for i, 15 for xvi, 16 for 1, up through 38 for page 23. With a little work, one could easily redefine TeX cross-reference and index macros to include file names and absolute page numbers, thereby implementing dynamic and automatic PDF linking. One method for keeping track of absolute page numbering in TeX is to define a new counter and increment it in the TeX output routine.

We can combine the methods of the two previous examples by having the TeX macro `PDFnamed-destination` include another TeX macro which writes each instance, named destination, and filename, to a global output file containing calls to each document file:

```
% Global file containing
% named destinations
\input document1.nd
\input document2.nd
   .
   .
\input documentN.nd
```

In this manner, all files can be accessed somewhat independently and named references for each document can be easily updated. By referencing this file, one can determine in which file a named destination occurs.

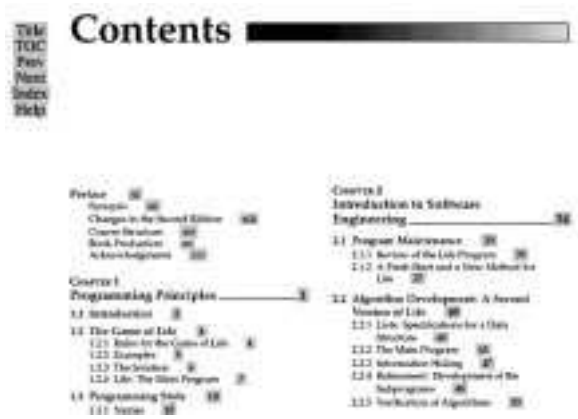


Figure 2: An example of a PDF document containing links from Table of Contents entries to their locations in the main body of the text.

Duplicate named destinations should not occur. Figures 2 and 3 show several examples of how this combination can be implemented to give PDF links. The small shaded boxes represent PDF link boxes; clicking on these boxes will jump the reader to the destination page.

Figure 3: An example of index entries containing links to their locations in the main body of the text.

executable should be thoroughly tested before being distributed.

Figure 4: An example of launching an application in a PDF file by clicking the shaded boxes containing the "D".

Launching applications from PDF links

One final useful example of a PDF link is to allow a user to launch an application program while reading the PDF document. The following code allows an executable to be opened directly and related files to be opened using associations:

```
\def\pdflaunchapp#1{{%
% #1 = filename (pathname is optional)
\special{verbatim=
" [ /Rect [ currentpoint
            2 neg add exch
           10 neg add exch
            currentpoint
            8 add exch
            0 add exch]
    /Border [0 0 0]
    /Action << /Type /Action
         /Subtype /Launch
         /File (#1) >>
    /Subtype /Link
    /ANN pdfmark "}}}
```

The TEX example

```
\pdflaunchapp{filename.c}
```

can be used to compile, or to open with an Integrated Development Environment (IDE), files having extensions associated with C compilers. Files such as MS-DOS executables can work, although these files can only be run in a DOS/Windows environment. The shaded boxes containing the uppercase "D" in Figure 4 indicates a PDF link to launch the program `compside.exe`. In general, this type of PDF link can be platform- and operating-system-specific; however, under certain conditions, it can be quite useful. Any link to an

Customized PDFlink radio buttons

We can now create documents which contain dynamically linked cross-references in the form of PDF links. We should now enhance our PDF documents by creating menus or user-defined buttons which will allow warping to different locations of the book. In Figures 1 to 4 we see a menu with such headings as **Title**, **TOC**, **Prev**, **Next**, **Index**, and **Help**; all these represent links to various locations in the book. These are inserted onto each page by including a TEX command in the output routine. The TEX output routine

```
\def\output{\shipout\vbox{\pdftoolbar
\makeheadline
\pagebody
\makefootline}%
\advancepageno
\global\advance\absolutepageno by 1
\ifnum\outputpenalty>-\@MM
\else\dosupereject\fi}
```

is taken from the set of macros used to create the pages in Figures 1 to 4. Note that there is also a reference to absolute page numbering in the sixth line, as we suggested earlier. The TEX macro for the PDF link toolbar

```
\def\pdftoolbar{{
\vbox to 0pt{\hbox to 0pt{\hskip -15pc
\vbox{\hsize=8pc%
\pdftitlepage
\pdftoc
\pdfprev
\pdfnext
\pdfindex
\pdfhelp
}\hss}\vss}}}
```

is quite simple. Depending on its implementation, however, must not affect the remainder of the page body. The TEX macro `\pdftoc`, for example, is simply a line containing a PDF link using the `\pdfxrffileopen` macros.

Acrobat 4

In this paper we have attempted to define a set of TEX macros which will create on-line PDF documents which are platform independent. Since we cannot foresee advancements in software development, with the release of Acrobat 4 and future versions, there remains the potential for better and more visually appealing on-line documents. The examples used in this paper are based on existing PDF features found in the on-line document *Portable Document Format Reference Manual, Version 1.2, November 1997,* by Adobe Systems. This edition is rather old, but one can get the most recent PDF version at the Adobe website `www.adobe.com`.

PreTEX and PDF

PreTEX is a preprocessor and macro package for TEX, developed by PreTEX, Inc., which supplies an author with many tools to simplify the writing and management of larger (book length) projects. (For a discussion of PreTEX, see the article by R. Kruse elsewhere in this issue.) One of PreTEX's important expansions of the resources available to an author is the inclusion of PDF links by using TEX macro definitions similar to the examples above. In this way, a book can be published both on paper and in an electronic form that incorporates automatically generated PDF links for cross-references, index entries, launching application programs, and the like. The current set of PDF macros are not stable enough; they will be made freely available in the spring of next year (1999).

Cheers.

References

Adobe Systems Incorporated. *PostScript Language Reference Manual*. Addison-Wesley, Reading, Massachusetts, 1993.

Adobe Systems Incorporated. *Portable Document Format Reference Manual*. Addison-Wesley, Reading, Massachusetts, 1993.

Hendrickson, Amy. "Real Life LATEX: Adventures of a TEX Consultant." *TUGboat* **19**(2), 162–167 (1998).

Kruse, Robert. "Managing Large Projects with PreTEX: A Preprocessor for TEX." 1999. (See elsewhere in these Proceedings.)

Merz, Thomaz. *PostScript and Acrobat/PDF* Springer-Verlag, Berlin, Heidelberg, 1997.

Taylor, Philip. "Computer Typesetting or Electronic Publishing? New trends in scientific publishing." *TUGboat* **17**(4), 367–381 (1996).

Thanh, Hàn Thế. "Improving TEX's Typeset Layout." *TUGboat* **19**(3), 284–288 (1998).

Thành, Hàn Thế, and Sebastian Rahtz. "The pdfTEX user manual." *TUGboat* **18**(4), 249–254 (1997).

A TEX Haiku

```
\expandafter\def
\csname def\endcsname
{\message{farewell}}\bye
```

—Sebastian Rahtz

A TEXnician's Haiku

This haiku for TEXnicians consists entirely of TEX control sequences; furthermore, it forms a valid TEX assignment statement—provided that a certain control sequence that is normally undefined is defined in an obvious way. *Can you identify the control sequence in question*?

```
\catcode\csname
\romannumeral\parshape
\endcsname\month
```

—Michael Downes

A TEX Cheer

```
Flush to the left
Flush to the right
\vskip, \hskip, type type type
```

Michael Sofka

A Web-Based Submission System for Meeting Abstracts

Hu Wang
American Institute of Physics
1NO1
2 Huntington Quadrangle
Melville, NY 11747
hwang@aip.org

Abstract

As one of the services provided to AIP's Member Societies, we publish program books and abstract books for meetings sponsored by the societies. In this paper, we describe a Web-based client-server abstract submission system. It uses HTML forms to gather and validate user input of abstracts and related information, provide on-line proof before officially submitting, and store the input in LaTeX files and a database. From the publishing production's point of view, the major benefit of such a system is the greatly improved quality of well-structured submissions, which makes it possible to streamline the whole production cycle. Some possibilities with its integration into a database publishing system are briefly discussed afterwards.

Introduction

The American Institute of Physics (AIP) publishes program books and abstract books for some of its member societies' meetings. The last few years have seen many changes in the ways some societies' meeting abstracts are submitted and published.

At the beginning, hardcopy abstracts prepared in various text formatting software were mailed in and published with the cut-and-paste method. It was non-digital, inefficient, and the resulting quality of finished products was understandably poor.

Then, with the popularity of LaTeX in scientific communities and the widespread availability of email, came email-based electronic submissions. Authors would download an appropriate LaTeX template file that included predefined tags (LaTeX commands and environments), fill it out, and email it to a designated address. A program would collect the submissions and save them as individual LaTeX files.

The email approach had the potential advantages that all files were standardized, well tagged, and the publishing process could be highly automated. Unfortunately, since there are still many authors who are not familiar with or do not have LaTeX, many submissions had to be manually corrected for syntax errors by the production staff, which was time consuming and sometimes impractical. This often led to syntactically invalid submissions, which in turn prohibited auto-processing. Another drawback was that the process was not interactive. As for authors, those without LaTeX could not proofread their abstracts, and figures could not be easily submitted with the abstracts.

We needed to develop a system that could take advantage of the potential strength of the email-based method while avoiding its shortcomings, namely, the non-interactiveness and lack of control on the quality of submissions.

A Web-based system

As the World Wide Web spreads around the world (for some society meetings, over one third of the contributed papers come from abroad), it lends itself naturally to a solution of our problems.

We have developed an automated, Web-based, client-server meeting abstract submission system with the following features:

- interactive user interfaces via HTML forms
- on-line LaTeX help info
- choice of LaTeX or non-LaTeX entry methods
- uploading figures with abstracts
- on-line proof viewing and syntax validation
- editable abstracts
- easy integration with publishing phases

Regardless of the input method chosen by the authors, each submission results in a well-tagged LaTeX file that is free of syntax errors, a valid HTML file into which the input data is injected, and updated database records.

LaTeX is used as the ultimate file format for the abstract collecting phase for two reasons: it fits

well with our existing production systems and it is a widely accepted standard for publishing scientific and technical documents. In fact, the on-line proof is produced by LaTeX, along with `dvips`, and some PostScript-to-gif conversion program.

System requirements

Client: Internet connection and a conventional Web browser.
Server: Web server, LaTeX, `dvips`, Image Alchemy or `ps2gif`, GhostScript, and CGI scripts.

In the following section, we describe the implementation of this system.

Implementation

A user accesses the system by using a Web browser to connect to a designated URL and entering the meeting password included in the calls for paper distributed to the society members.

Once logged in, the user indicates whether he or she is entering a new abstract or wishes to edit a previously submitted abstract by clicking on the appropriate radio button. The former choice prompts the author to indicate, via radio buttons, the total number of mailing addresses needed for all the authors of the abstract and the total number of figures (up to 2) in the abstract. An appropriate template is then displayed for the user to fill out. The latter choice requires the user to enter the abstract number and the PIN that were both assigned and emailed to the author by the system when the abstract was initially submitted. In this case, their filled-out template will be shown for editing. Authors who forget their abstracts numbers and PINs can query the system and the results will be emailed back to them.

Template. One of the key components of the system is the Web template. The elements from this HTML form are captured for insertion into the database and are also tagged for the LaTeX file that results from completing and submitting the template form.

To accommodate users who are unfamiliar with or do not need LaTeX, the top of the template has two radio buttons labeled "Straight Text" (i.e. non-LaTeX) and "LaTeX", respectively, for choosing the method of entry. With the former method, no LaTeX commands are recognized because everything entered will be treated literally, which means it is impossible to choose fonts or set math expressions. The usual LaTeX commands are allowed with the LaTeX entry method — except for `\thanks`, `\footnote`, `\begin{center}` and the like.

The template has hyperlinks to sample inputs for both entry methods, and to LaTeX help on how to input symbols and mathematical expressions. When clicked, these links open separate windows for easy reference.

The following form elements are used in the template:

- radio buttons for choosing the entry method
- presenting author's name
- corresponding author's name, address, country, phone, email address, fax
- title and short title
- author's name and address
- `<textarea>` for abstract body
- file selection fields for uploading figures (if any)
- figure caption (if any)
- topics of paper
- requested presentation method
- hidden fields

Hidden fields are for passing state variables information from one invocation of a CGI script to the next. They are embedded in the template to identify the client session, to indicate if it's a new or previously submitted abstract, as well as the number of figures and author groups in the abstract.

CGI scripts. Processing of submitted forms is handled by CGI scripts, which are programs communicating (via the Web server) with clients. The essential scripts are those that deal with filled templates and proof screens. They perform the following tasks sequentially:

1. Validate the form and figures (PostScript or Tiff only, if any). This checks if any required input fields are empty, if illegal input is found (e.g. no TeX or LaTeX commands are allowed in the Straight Text entry method, `\thanks` is not allowed in either method, and so on.), and if the uploaded figure is a valid PostScript file (the PostScript language interpreter Ghostscript is used for this purpose) or a Tiff file.

   Any input that fails to pass the validation will cause an appropriate error message to be displayed.

   Here, JavaScript, which is faster on the client side, can be used for form validation also.
2. Build a temporary LaTeX file and process it. If there is no syntax error, run `dvips` and Alchemy to generate the gif image and display the image to the user for proof. If there are syntax errors, extract the error message from the log

file, display it, and ask the user to go back to the template screen and fix the LaTeX error.

We choose the gif format for proofing because it is accessible without requiring browser plug-ins or help applications. Of course, the PDF format may be used for proofing, which requires the Adobe Acrobat Reader for viewing.

In order to show the user the dynamically generated gif, we must prevent browser caching from displaying the old gif. This can be done by proper HTTP headers such as 'Cache-Control: no-cache' or 'Cache-Control: no-store'. The gif file for a typical abstract is about 20Kb in size.

Building LaTeX files via the Straight Text method warrants a little note here, as whatever the user has entered will be treated as literal. The CGI script must handle the following TeX special characters carefully to properly escape them:

```
# $ % & _ { } ~ ^ \ < > |
```

For example, author's input `<` should be converted to `$<$` in the LaTeX file; otherwise, unexpected output will result even though LaTeX does not complain.

It's rather straightforward to build LaTeX files with the LaTeX entry method — just insert the author input data into the arguments of appropriate LaTeX control sequences. Not surprisingly, the LaTeX file's tags correspond nicely to the template's elements. For instance, these tags are used:

- `\pauthor`
- `\cauthor`
- `\caddress`
- `\cphone`
- `\cemailaddr`
- `\cfax`
- `\title`
- `\author`
- `\authaddress`
- `\begin{abstract}`
- `\caption`
- `\category`
- `\pmethod`

Let us now demonstrate the relationship between author input in each entry method and the resulting LaTeX file.

The Straight Text method. In this sample, the user faces a screen with the prompt "Title" on the left and a blank box. The title text is input literally, like this:

Title: `Straight Text Mode #$% Title`

which will be converted into the following in the resulting LaTeX file:

```
\title{Straight Text Mode \#\$\% Title}
```

Notice that the literal input of the special characters `#$%` has been correctly converted with the addition of backslashes.

The LaTeX method. By selecting this method, authors can directly input LaTeX code. In this sample, the author is facing a similar screen, with "Presenting Author" as the prompt to the left of the boxed area for the name:

First name: `P\'al` *Last name:* `Kn\"oll`

will result in this statement in the LaTeX file:

```
\pauthor{P\'al}{Kn\"oll}
```

3. Build a temporary HTML file into which the user-input data is inserted. This file will be needed if the user wants to edit a previously submitted abstract later on.

   Here, care is needed too. First, every `"` character entered by the user must be replaced in the resulting HTML file by the *entitized* version, i.e. by `&quot;` so that it does not interfere with the HTML form's element value delimiter `"`. Second, any scrolling list element in the template must be placed in the HTML file and a `SELECTED` attribute inserted inside the `<option>` that has been selected by the user.

   For instance, the previous example of entering the presenting author name would yield the following lines in the HTML file:

```
First name:<input name=p_fname
           type=text value="P\'al">
Last name:<input name=p_lname
     type=text value="Kn\&quot;oll">
```

   Please note, when displaying an HTML file, the browser replaces all entity references such as the above one with the corresponding characters. Therefore, the CGI script that processes the submitted form does not need to *de-entitize* any form data.

4. Insert relevant information into the database. This may be needed later for searching, reporting, mailing label printing, etc.

   After the LaTeX source file is error-free and a proof of its output is displayed, the user may decide to finally submit it to the system or may go back to the template to massage it further

and then go through the proof and final submission cycle again.

5. Final submit. For a new submission, assign a new sequential abstract number; for a re-submission, extract its abstract number from the corresponding hidden field. In either case, rename the source file, the HTML file, and any figure files to the abstract number with proper extensions, email the abstract number and a randomly generated PIN to the corresponding email address, and insert or update the database records accordingly.

Conclusions

This system offers many benefits to both users and the production staff.

Users' benefits. Self-evident HTML form templates, choice of entry methods, easy figure handling, LaTeX syntax validation (this could be a mixed bag for LaTeX newbies because the error messages may be too cryptic for them), on-line proof viewing, and editing previously submitted abstracts.

Production benefits. Since the rigorous validation processes shift more responsibilities to the users, the abstract submission quality is greatly improved. The production staff now start the game with standardized, well-structured data, which makes it possible for the subsequent events to be highly automated. In fact, this system can be expanded into a comprehensive database publishing system for handing the following conferences-related tasks:

- During the period of collecting abstract submissions, set up a cron job for the conference coordinator to print out abstracts and accompanying figures received the previous day, to generate a sort-by-category and sort-by-presenting-author list of abstracts received so far.
- For program committee use: abstract database search for various fields.
- Insert into the database the program committee's decisions of acceptance and rejection, as well as the meeting sessions information (such as the session name, schedule, location, chairperson, invited talks, contributed talks, posters, time slot for each presentation, etc.).
- Generate letters of acceptance and rejection, as well as mailing labels.
- Generate program books and abstract books.
- Searchable on-line program listing and viewing.

Acknowledgements

Several people at the AIP were involved in this project: Don Lang, Chris Hamlin, and Kevin McGrath. My thanks to all of them — especially Chris Hamlin, from whom I have learned many TeXniques.

I would also like to thank the TUG99 reviewers and the Proceedings editor, Christina Thiele, for their constructive suggestions and comments.

Hyphenation on Demand

Petr Sojka
Faculty of Informatics
Masaryk University Brno
Botanická 68a, 602 00 Brno
Czech Republic
sojka@informatics.muni.cz

Abstract

The need to fully automate the batch typesetting process increases with the use of TEX as the engine for high-volume and on-the-fly typeset documents which, in turn, leads to the need for programmable hyphenation and line-breaking of the highest quality.

An overview of approaches for building *custom* hyphenation patterns is provided, along with examples. A methodology of the process is given, combining different approaches: one based on morphology and hand-made patterns, and one based on word lists and the program PATGEN. The method aims at modular, easily maintainable, efficient, and portable hyphenation. The bag of tricks used in the process to develop custom hyphenation is described.

Motivation

> *In principle, whether to hyphenate or not is a style question and CSS [Cascading Style Sheets] should develop properties to control hyphenation. In practice, however, for most languages there is no algorithm or dictionary that gives all (and only) correct word breaks, so some help from the author may occasionally be needed.*
> — (Bos, 1999)

Separation of content and presentation in today's open information managment style in the sense of SGML/XML (Goldfarb, 1990; Megginson, 1998) is a challenge for TEX as a batch typesetting tool. The attempts to bring TEX's engine to untangle presentation problems in the WWW arena are numerous (Sutor and Díaz, 1998; Skoupý, 1998).

One bottleneck in the high-volume quality publishing is the proofreading stage — line-breaking and hyphenation handling that need to be fine tuned to the layout of particular publication. Tight deadlines in paper-based document production and high-volume electronic publishing put additional demands for better automation of the typesetting process. The need for multiple presentations of the same data (e.g., for paper and screen) adds another dimension to the problem. Problems with hyphenation are often one of the most difficult. As most TEX users are perfectionists, fixing and tuning hyphenation for every presentation is a tedious, time-consuming task.

We have already dealt with several issues related to hyphenation in TEX (Sojka and Ševeček, 1995; Sojka, 1995). On the basis of our being involved in typesetting tens of thousands of TEX pages of multilingual documents (mostly dictionaries), we want to point out several methods suitable for the development of hyphenation patterns.

Pattern generation

> *There is no place in the world that is linguistically homogeneous, despite the claims of the nationalists around the world.*
> — (Plaice, 1998)

Liang (1983), in his thesis written under Knuth's supervision, developed a general method to solve the hyphenation problem that was adopted in TEX82 (Knuth, 1986a, App. H). He wrote the PATGEN program (Liang and Breitenlohner, 1999), which takes

- a list of already hyphenated words (if any),
- a set of patterns (if any) that describes "rules",
- a list of parameters for the pattern generation process,
- a character code translation file (added in PATGEN 2.1; for details see Haralambous (1994),

and generates

- an enriched set of patterns that "covers" all hyphenation points in the given input word list,
- a word list hyphenated with the enriched set of patterns (optional).

The patterns are loaded into TEX's memory and stored in a data structure (cf. Knuth, 1986b, parts 40 – 43), which is also efficient for retrieval — a variant of *trie* memory (cf. Knuth, 1998, pp. 492 – 512). This data structure allows hyphenation pattern searching in linear time with respect to the pattern length. The algorithm using a trie that "outputs" possible hyphenation positions may be viewed as finite automaton with output (Mealy automaton or transducer).

Pattern development

> *. . . problems [with hyphenation] have more or less disappeared, and I've learnt that this is only because, nowadays,* every *hyphenation in the newspaper is* manually checked *by human proof-readers.*
> — (Jarnefors, 1995)

Studying patterns that are available for various languages shows that PATGEN has only been used for about half of the hyphenation pattern files on CTAN (cf. Table 1 in Sojka and Ševeček, 1995).

There are two approaches to hyphenation pattern development, depending on user preferences. Single authors using TEX as an authoring tool want to minimize system changes and want TEX to behave as a fixed point so that re-typesetting of old articles is easily done, thanks to backwards compatibility. For such users, one set of patterns that is fixed once and for all might be sufficient.

On the other hand, for publishers and corporate users with high-volume output, it is more efficient to make a long-term investment into development of hyphenation patterns for particular purposes. I remember one TEX user saying that my suggestion to enhance standard hyphenation patterns with custom-made ones to allow better hyphenation of chemical formulæ would save his employer thousands of pounds per year. Of course, with this approach, one has to archive *full* sources for every publication, together with hyphenation patterns and exceptions.

One of the possible reasons PATGEN has not been used more extensively may be the high investment needed to create hyphenated lists of words, or better, a morphological database of a given language.

Pattern bootstrapping and iterative development

> *The road to wisdom?*
> *Well it's plain and simple to express:*
> *Err and err and err again*
> *but less and less and less.*
> — (Hein, 1966)

When developing new patterns, it is good to start with the following bootstrapping technique with iteration, which should avoid the tedious task of manually marking hyphenation points in huge lists of words:

1. Write down the most obvious initial patterns, if any, and/or collect "the closest" ones (e.g., consonant-vowel rules).
2. Extract a small word list for the given language.
3. Hyphenate current word list with current set patterns.
4. Check all hyphenated words and correct them; in the case of errors return to step 3.
5. Collect a bigger word list.
6. Use the previously generated set of patterns to hyphenate the words in this bigger list.
7. Check hyphenated words, and if there are no errors, move to step 9.
8. Correct word list and return to step 6.
9. Generate final patterns with PATGEN with parameters fitted for the particular purpose (tuned for space or efficiency).
10. Merge/combine new patterns with other modules of patterns to fit the particular publishing project.

To find an initial set of patterns, some basic rules of hyphenation in the specific language should be known. Language can be grouped into one of two categories: those that derive hyphenation points according to etymology and those that derive hyphenation according to pronunciation — "syllable-based" hyphenation. For the first group of languages, one should start with patterns for most frequent endings and suffixes and prefixes. For syllable-based hyphenation, patterns based on sequences of consonants and vowels might be used (cf. Chicago Manual of Style, 1993, Section 6.44, and Haralambous, 1999) as first approximation of hyphenation patterns.

As using TEX itself for hyphenation of word lists and development of patterns may be preferred to other possibilities, we will start with this portable solution, using hyphenation of phonetic transcriptions as an example of a syllable-based "language".

Let's start with some plain TEX code to define consonant-vowel (CV) patterns:

```
% ... loading plain.tex
%     without hyphen.tex patterns ...
\patterns{cv1cv cv2c1c ccv1c cccv1c
ccccv1c cccccv1c v2v1 v2v2v1 v2v2v2v1
...
}
```

There is a way to typeset words together with their hyphenation points in TEX; the code from Olšák (1997, with minor modifications) looks like this:

```
\def\showhyphenspar{\begingroup
 \overfullrule=0pt \parindent0pt
 \hbadness=10000 \tt
 \def\par{\setparams\endgraf\composelines}%
 \setbox0=\vbox\bgroup
            \noindent\hskip0pt\relax}

\def\setparams{\leftskip=0pt
 \rightskip=0pt plus 1fil
 \linepenalty1000 \pretolerance=-1
 \hyphenpenalty=-10000}

\def\composelines{%
 \global\setbox1=\hbox{}%
 \loop
   \setbox0=\lastbox \unskip \unpenalty
   \ifhbox0 %
     \global\setbox1=\hbox{%
       \unhbox0\unskip\hskip0pt\unhbox1}%
 \repeat
 \egroup % close \setbox0=\vbox
 \exhyphenpenalty=10000%
 \emergencystretch=4em%
 \unhbox1\endgraf
 \endgroup}
```

Now, we will typeset our word list in the typewriter font without ligatures. To use the CV patterns defined above we need to map word characters properly:

```
% vowels mapping
\lccode`\a=`v  \lccode`\e=`v
\lccode`\i=`v  \lccode`\o=`v
...
% consonants
\lccode`\b=`c  \lccode`\c=`c
\lccode`\d=`c  \lccode`\f=`c
...
\raggedbottom \nopagenumbers
\showhyphenspar
The need to fully automate the
batch typesetting process increases
with the use of word in wordlist
...
\par\bye
```

Finally, extracting hyphenated words from `dvi` the file via the `dvitype` program, we get our word list hyphenated by our simple CV patterns.

Another way to get the initial word list hyphenated is to use `PATGEN` with initial patterns and no new level, letting `PATGEN` hyphenate the word list that was input. PERL addicts may want to use the PERL hyphenation module (Pazdziora, 1997) for the task.

Once the job of proofreading the word list is finished, we can generate new patterns and collect other words in the language. Using new patterns on the new collection will show the efficiency of the process.

Fine tuning of patterns may be iterated, once `PATGEN` parameters are set, so that nearly 100 % coverage of hyphenation points is achieved in every iteration. The setting of such `PATGEN` parameters may be difficult to find on the first attempt. Setting of these parameters is discussed in Sojka and Ševeček (1995).

Modularity of patterns

It is tractable for some languages to create patterns by hand, simply by writing patterns according to the rules for a given language. This approach is, however, doomed to failure for complex languages with several levels of exceptions. Nevertheless, there are special cases in which we may build pattern modules and concatenate patterns to achieve special purpose behaviour. This applies especially when additional characters (not handled when patterns have been built originally) may occur in words that we still want to hyphenate.

Patterns generated by Raichle (1997) may serve as an example that can be used with any fonts in the standard LATEX eight-bit T1 font encoding, to allow hyphenation after an explicit hyphen. Similar pattern modules can be written for words or chemical formulæ that contain braces and parentheses. These can be combined with "standard" patterns in the needed encodings. Some problems might be caused by the fact that TEX does not allow metrics to be defined for `\lefthyphenmin` and `\righthyphenmin` properly — we might want to say that ligatures, for instance, count as a single letter only or that some characters should not affect hyphenation at all (e.g. parentheses in words like colo(u)r). We must wait until some naming mechanisms for output glyphs (characters) is adopted by the TEX community for handling these issues.

Adding a new primitive for the hyphenmin code — let's call it `\hccode`, a calque on `\lccode` — would cause similar problems: changing it in mid-paragraph would have unpredictable results.[1]

It is advisable to create modules or libraries of special-purpose hyphenation patterns, such as the

[1] ε-TEX v2 has a new feature to fix the `\lccode` values during the pattern read phase.

ones mentioned above, to ease the task of pattern development. These patterns might be written in such as as to be easily adaptable for use with core patterns of a different language.

Common patterns for more languages

Having large hyphenated word lists of several languages the possibility then exists to make multilingual or special-purpose patterns from collections of words by using `PATGEN`. Joining word lists and generating patterns on demand for particular publications is especially useful when the word databases are structured and split into sublists of personal names, geographic names, abbreviations, etc. These patterns are requested when typesetting material in which language switching is not properly done (e.g. on the WWW).

Czech and Slovak are very closely related languages. Although they do not share exactly the same alphabet, rules for hyphenation are similar. That has led us to the idea of making one set of hyphenation patterns to work for both languages, saving on space in a format file that supports both. In the Czech/Slovak standard TEX distribution there is support for different font encodings. For every encoding, hyphenation patterns have to be loaded as there is no character remapping on the level of trie possible. Such Czechoslovak patterns would save patterns for each encoding in use.

It should be mentioned that this approach cannot be taken for any set of languages as there may be, in general, identical words that hyphenate differently in different languages; thus, simply merging word lists to feed `PATGEN` is not sufficient without degrading the performance of patterns by forbidding hyphenation in these conflicting words (e.g. re-cord vs. rec-ord).

Phonetic hyphenation

As an example of custom-made hyphenation patterns, the patterns required to hyphenate a phonetic (IPA) transcription are described in this section. Dictionaries use this extensively — see Fig. 1,[2] taken from Kirsteinová.

The steps used to develop the hyphenation patterns for this dictionary were similar to those described in the previous section on bootstrapping:

1. Write down the most obvious (syllable) patterns.
2. Extract all phonetic words from available texts.

[2] The IPA font used is TechPhonetic, downloadable from `http://www.sil.org/ftp/PUB/SOFTWARE/WIN/FONTS/`.

akkompagnement *sb* [ækʌmpænjə-ˈmaŋ] *-et, -er* hudební doprovod *m*
alimentationsbidrag *sb* [ælimɛntæ-ˈšo:'nsˌbiˌdra:'w] *-et,* - alimenty *pl*, příspěvek *m* na výživné dítěte
befolknings‖eksplosion *sb* [beˈfʌl'g-neŋs-] *-en, -er* populační exploze *f* ■ **-tilvækst** *-en, -er* přírůstek *m* obyvatelstva ■ **-tæthed** *-en, -er* hustota *f* obyvatelstva
bemærkelsesværdig *adj* [beˈmæʀ-gəlsəsˌvæʀ'di] *-t, -e* pozoruhodný
beslutningsdygtig *adj* [beˈslud-neŋsˌdøgdi] *-t, -e* schopný rozhodovat; **den lovgivende forsamling var ~** zákonodárné shromáždění bylo schopné se usnášet

Figure 1: Example of phonetic hyphenation in Kirsteinová and Borg (1999).

3. Hyphenate this word list with the initial set of patterns.
4. Check and correct all hyphenated words.
5. Generate final quality patterns.

In bigger publishing projects efforts like this pay off very quickly.

Hyphenation for an etymological dictionary

In some publications (Rejzek, in prep., for example), a different problem can arise: the possibility of having more than 256 characters used within a single paragraph. This problem cannot, in general, be easily solved[3] within the frame of TEX82. We thus tried Ω, the typesetting system by Plaice and Haralambous, for this purpose. One has to create special virtual fonts (e.g., by using the fontinst package) on top of the Ω ones, in order to typeset it — see Fig. 2.

More hyphenation classes

> *But at least I can point out a minor weakness of TEX's algorithm: all possible hyphenations have the same penalty. This might be ok for english, but for languages like German that have a lot of composite words there should be the ability to assign lower penalties between parts of a composite i.e. Um-brechen should be favored against Umbre-chen.*
> — (Hars, 1999)

[3] One could try to re-encode all fonts used in parallel in some paragraph such that they share the same `\lccode` mappings, but this exercise would have to be made for each multilingual-intensive publication, again and again.

naivní ‘prostoduchý, dětinský’, *naivita, naivka.* Přes něm. *naiv* z fr. *nai:f* tv., původně ‘přirozený, opravdový’ z lat. *nātīvus* ‘přirozený’ od *nātus,* což je příč. trp. od *nāscī* ‘rodit se’. Srov. ↑*nacionále.*

náruživý ‘vášnivý, silně zaujatý’, *náružívost.* Jen č. Souvisí s č.st. *oruží* ‘zbroj, zbraň, náčiní’ (všesl.). Psl. kořen **-rǫg- (B1, B7)* nejspíš souvisí s lit. *įrangùs* ‘prudký’ (HK), *rángtis* ‘spěchat’, *ren~gtis* ‘chystat se’, *rangà* ‘přístroj, nástroj’, dále asi se střhn. *ranc* ‘rychlý, vířivý pohyb’, něm. *renken* ‘pohybovat se krouživě sem tam’, angl. *wrench* ‘trhnout, vykroutit’, vše by to bylo od ie. **u̯reng-* ‘kroutit, ohýbat’ a vzdáleně příbuzné by bylo ↓*vrhat.*

nebozez ‘vrták na dřevo’, *nebozízek.* Hl. *njeboz,* sln. *nabôzec.* Přejato z germ., forma by ukazovala až na germ. **naba-gaiza* před změnou *-z->-r- (A5, B1),* která už je provedena v sthn. *nabagēr* (něm. *Näber* tv.). První část germ. slova odpovídá něm. *Nabe* (viz ↑*náboj*), druhá něm.st. *Ger* z gót. **gaiza-* ‘něco špičatého’.

Figure 2: Using Ω to typeset paragraphs in which words from languages with more than 256 different characters may appear and be hyphenated in parallel.

Some suggestions on handling multiple hyphenation classes were suggested in Sojka (1995). A prototype implementation of ε-TEX and `PATGEN` has recently been done (Classen, 1998). For wider adoption of such improvements availability of large word lists and development of new patterns is crucial. Many of the methods mentioned above could be used to develop such multi-class/multi-purpose patterns. Allen (1990) contains such a word list, which shows that some publishers do pay attention to linebreaking details.

Speed considerations

Even though hyphenation searches using a trie data structure is fast, searching for unnecessary hyphenation points is a waste of time. It is advisable to tell TEX where words shouldn't be hyphenated. Comparing several possibilities for suppressing hyphenation, the option of setting `\lefthyphenmin` to 65 is slightly faster than switching to `\language`, which has no patterns. These solutions outperform the `\hyphenpenalty 10000` solution by a fair amount (cf. Arsenau, 1994).

Reuse of patterns

Sometimes we need the same patterns with different `\lefthyphenmin` and `\righthyphenmin` parameters. The suggested approach is not to limit hyphens close to word boundaries during the pattern generation phase but to use TEX's `\setlanguage` primitive. This can be done to achieve special hyphenation handling for the last word in a paragraph (e.g., a higher `\righthyphenmin`) given proper markup by a preprocessing filter. For example:

```
\newcount\tmpcount
\def\lastwordinpar#1{%
\tmpcount=\righthyphenmin
\righthyphenmin5
\setlanguage\language #1
\expandafter\righthyphenmin\the\tmpcount
\setlanguage\language}

\showhyphens{demand}
\lastwordinpar{demand\showhyphens{demand}}
\bye
```

Future work

> *If you find that you're spending almost all your time on practice, start turning some attention to theoretical things; it will improve your practice.*
> — (Knuth, 1989)

It seems inevitable that embedding of language-specific support modules will be necessary for *the* typesetting system in the future. These demands may not only apply for hyphenation but also for spelling or even grammar checkers. As even people using WYSIWYG systems may use tools that help to visualise possible typos (in color, etc.) on the fly, the computing power of today's machines is surely sufficient to do the same in batch processing with even better results.

The idea of using patterns to capture mappings specific for particular languages or dialect modules can be further generalized for different purposes and mappings. The use of the theory of finite-state transducers (Mohri, 1996; Mohri, 1997; Roche and Schabes, 1996) to implement other classes of language modules looks promising.

Summary

Some computerized typesetting methods in frequent use today may render a conservative approach to word division impractical. Compromise may therefore be necessary pending the development of more sophisticated technology.
— Chicago Manual of Style (1993, Section 6.43)

We have outlined some of the possibilities offered by TEX and PATGEN for the development of customized hyphenation patterns. We have suggested bootstrapping and iterative techniques to facilitate pattern development. We also suggest wider employment of PATGEN and preparation of hyphenated word lists and modules of patterns for easy preparation of hyphenation patterns on demand in today's age of digital typography (Knuth, 1999).

Acknowledgements. We thank Bernd Raichle for valuable comments and corrections to the paper. We are indebted to the Proceedings editor for wording improvements. The presentation of this work has been made possible through support from the Ministery of Education, Youth and Physical Training (MŠMT ČR grant VS97028).

References

Allen, R.E. *The Oxford Spelling Dictionary*, volume II of *The Oxford Library of English Usage*. Oxford University Press, 1990.

Arsenau, Donald. "Benchmarking paragraphs without hyphenation". Posting to the Usenet group news:comp.text.tex on Dec 13, 1994.

Bos, Bert. "Internationalization / Localization". http://www.w3.org/International/O-HTML-hyphenation.html, 1999.

Chicago Manual of Style. *The Chicago Manual of Style*, 14th edition, 1993.

Classen, Matthias. "An extension of TEX's hyphenation algorithm". ftp://peano.mathematik.uni-freiburg.de/pub/etex/hyphenation/, 1998.

Goldfarb, Charles F. *The SGML Handbook*. Clarendon Press, Oxford, 1990.

Haralambous, Yannis. "A Small Tutorial on the Multilingual Features of PATGEN2". In electronic form, available from CTAN as info/patgen2.tutorial, 1994.

Haralambous, Yannis. "From Unicode to Typography, A Case Study: The Greek Script". Proceedings of 14th International Unicode Conference, preprint available from http://genepi.louis-jean.com/omega/boston99.pdf, 1999.

Hars, Florian. "Typo-l email discussion list". 1999.

Hein, Piet. *Grooks*. MIT Press, Cambridge, Massachusetts, 1966.

Jarnefors, Olle. "ISO-10646 email discussion list". 1995.

Kirsteinová, Blanka and B. Borg. *Dánsko-český slovník, Dansk-Tjekkisk Ordbog [Danish-Czech dictionary]*. LEDA, Prague, Czech Republic, 1999.

Knuth, Donald E. *The TEXbook*, volume A of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA, 1986a.

Knuth, Donald E. *TEX: The Program*, volume B of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA, 1986b.

Knuth, Donald E. "Theory and Practice". Keynote address for the 11th World Computer Congress (Information Processing '89), 1989.

Knuth, Donald E. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley, 1998.

Knuth, Donald E. *Digital Typography*. CSLI Lecture Notes 78. Center for the Study of Language and Information, Stanford, California, 1999.

Liang, Frank. *Word Hy-phen-a-tion by Com-put-er*. Ph.D. thesis, Department of Computer Science, Stanford University, 1983.

Liang, Frank and P. Breitenlohner. "PATtern GENeration Program for the TEX82 Hyphenator". Electronic documentation of PATGEN program version 2.3 from web2c distribution on CTAN, 1999.

Megginson, David. *Structuring XML Documents*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1998.

Mohri, Mehryar. "On some applications of finite-state automata theory to natural language processing". *Natural Language Engineering* **2**(1), 61–80, 1996.

Mohri, Mehryar. "Finite-State Transducers in Language and Speech Processing". *Computational Linguistics* **23**(2), 269–311, 1997.

Olšák, Petr. *TEXbook naruby [TEXbook topsy-turvy]*. Konvoj, Brno, 1997.

Pazdziora, Jan. "TeX::Hyphen — hyphenate words using TEX's patterns". CPAN: modules/by-authors/Jan_Pazdziora/TeX-Hyphen-0.10.tar.gz, 1997.

Plaice, John. "pdftex email discussion list". http://www.tug.org/archives/pdftex/msg01913.html, 1998.

Raichle, Bernd. "Hyphenation patterns for words containing explicit hyphens". CTAN/language/hyphenation/hypht1.tex, 1997.

Rejzek, Jan. *Etymologický slovník českého jazyka [Czech Etymological Dictionary]*. LEDA, Prague, Czech Republic, in prep..

Roche, Emmanuel and Y. Schabes. *Finite-State Language Processing*. MIT Press, 1996.

Skoupý, Karel. "$\mathcal{NTS}$: A New Typesetting System". *TUGboat* **18**(3), 318–322, 1998.

Sojka, Petr. "Notes on Compound Word Hyphenation in TEX". *TUGboat* **16**(3), 290–297, 1995.

Sojka, Petr and P. Ševeček. "Hyphenation in TEX — Quo Vadis?". *TUGboat* **16**(3), 280–289, 1995.

Sutor, Robert S. and A. L. Díaz. "IBM techplorer: Scientific Publishing for the Internet". *Cahiers Gutenberg* **28–29**, 295–308, 1998.

The Young Man of Vancouver

There was a young man of Vancouver
who thought he admired Anita Hoover
but he looked at some macros
which ran under Windows
and now all he can think of is `\over`s

—Sebastian Rahtz

The TUG conference

Down the TEXing path we go
with a Sparc its not so slow
Up the network nodes we run
`\href` links can be so much fun
Round the browser wars we dodge
Sans MathML—a real hodge-podge
Home at last—the Web is **fast**—we wait for LATEX3
While Frank and David trade ideas,
Chris seeks terminology

—Christina Thiele

The Young Lady of Stanford

There was a young lady from Stanford
who delighted to play with Mac Word
she met a Don Knuth
who told her the truth
and now what she enjoys is absurd

—Sebastian Rahtz and Patrick Ion

Active TEX and the DOT Input Syntax

Jonathan Fine
Active TEX
203 Coldhams Lane
Cambridge, CB1 3HY
United Kingdom
fine@active-tex.demon.co.uk
http://www.active-tex.demon.co.uk

Abstract

The usual category codes give TEX its familiar backslash and braces input syntax. With Active TEX, all characters are active. This gives the macro programmer complete freedom in defining the input syntax. It also provides a powerful programming environment.

The DOT input syntax, like TROFF, uses a period at the start of the line as an escape character. However, its underlying element, attribute and content structure is based on SGML. It is both easy to use and easy to program for.

Conversion to other formats, such as SGML, HTML and XML, or to proprietary formats such as Word and RTF, will be straightforward. This is because the DOT syntax is rigorous. This new syntax will be described and demonstrated.

All manner of problems connected with TEX disappear when Active TEX packages are used. For example, all input errors can be detected and corrected before they can cause a TEX error message. This will make TEX accessible to many more users.

Visit http://www.active-tex.demon.co.uk for information and macros.

Introduction

Much has changed since the introduction of TEX in 1982. Computers have become cheaper, more plentiful and more powerful. The Internet has grown from a tool used largely by North American academics to become a mass medium subject to powerful commercial interests. And Microsoft, who supplied an operating system for IBM's first PC, has become a colossus.

Donald Knuth gave us a powerful and reliable typesetting system. Other systems may be easier to use and have all sorts of useful (and perhaps not so useful) features, but when it comes to the typographic quality of the resulting pages, TEX is still superior in many important respects to all of its competitors. No other software even comes close to matching it on its own home ground, which is technical books, articles and preprints that have large amounts of mathematical material.

Both individuals and publishers are now making information available on the Internet. This imposes new demands on the typesetting process. For many users, HTML (and perhaps soon its replacement, XML) is the preferred means for supplying and receiving textual material. Twenty years ago the typeset page was the principal result of the typesetting process. Today, users are wanting both typeset pages and HTML or similar pages. By typeset pages I mean both pages for printing in the usual way, and also pages for display, say in the Portable Document Format introduced by Adobe. (In principle, this term also includes the formatting of, for example, HTML, for display in a browser.)

Most TEX authors use a text editor (such as emacs) to prepare a computer file in the LATEX syntax, for example. Other authors will use a word processor to create a file that is stored in a proprietary format. Later down the line, these files will be typeset, converted into HTML and so forth.

By and large, the closer is the syntax of the author's file to being rigorous and compatible with the processing that will be applied to it, the better will be the outcome. Compromise may be necessary. With TEX each author became his or her own typesetter. Very often (LA)TEX files contain macro definitions, introduced for the author's own convenience. These definitions can be a great nuisance for those who have to deal with the file later,

particularly if they reside in an external file that becomes separated from the main manuscript.

This then is the present context for the use of TEX. Most TEX users now use the LATEX macro package, together with style files and additional packages. LATEX was developed in the early 1980s. The first edition of its manual was published in 1985, about a year after *The TEXbook*. It did a tremendous job of making the resources of TEX available to non-experts. Around 1990, however, its limitations became clear, and more than an inconvenience. One response was the birth of the LATEX3 project. In 1994 this group released LATEX2e. This helped to standardise the current situation.

Recently, Rahtz [11] described LATEX as "hugely powerful, but chaotic, and on the verge of becoming unmanageable." He also tells us that the CONTEXT macro package, due to Hans Hagen and Pragma, addresses this problem by incorporating into itself "all the facilities you need." It does away with document classes and user-contributed packages.

Plain TEX, LATEX and CONTEXT all use the familiar 'backslash and braces' input syntax. This can cause problems, because it is not rigorous. Translation to HTML, for example, requires that the source document be parsed. But LATEX, for instance, is in general the only program that can successfully parse LATEX documents. This tends to result in (LA)TEX living in a world of its own, isolated from the world of desktop publishing and word processing. For some communities of users, such as mathematicians, this may not be a hardship.

Active TEX is a new way of using TEX. It allows us to either avoid or solve many of our problems. For the technical, its key idea is that each character is active, and is defined to be a macro. For example, the active letter '`a`' is a macro that expands to the control sequence `lcletter`, followed by an active '`a`'. Uppercase letters, digits and visible symbols are treated in a similar manner. By manipulating these definitions, we can make TEX do whatever we want. In particular, we can choose our input syntax. Both TEX and the system macro programmer work harder, to ease the life of both the user and the application programmer.

We will consider the problems relating to macros under three heads, namely Input Syntax, Macro Programming, and the Processing of Text. The final section gives the history and prospects of Active TEX. This article is somewhat informal, and should not be read as a definitive or legally binding statement. The software is still under development.

The DOT input syntax

There are two aspects to an input syntax, namely the concrete and the abstract. The abstract syntax is the structure or organisation that the syntax provides. The concrete syntax is a means of expressing objects so organised. Provided they have the same abstract syntax, translation from one concrete syntax to another will be a routine matter. The parsing process starts with a concrete instance of the structure, and produces from it events that characterise its abstract structure.

In SGML the concept of the content model provides a large part of the abstract syntax. It might say, for example, that an article such as this one consists of front matter, sections and end matter. Each section would be a sequence of paragraphs, together with figures and tables. The end matter might consist of appendices and a bibliography. The latter would be a sequence of bibliographic items.

In LATEX one would write

```
\section{Input syntax}
```

to start a section. In SGML one might write

```
<section title="Input syntax">
```

to start a section. This gives two examples of a concrete syntax. In SGML the `title` is an *attribute* of the `section` tag. In LATEX, `Input syntax` is a *parameter* of `\section`.

The abstract syntax provided by SGML is solid and well-understood. It is already widely used in data processing. The concrete syntax, however, tends to be somewhat verbose and difficult to use without dedicated software. This has been an obstacle to its widespread use. In the author's view, with XML this problem will become more acute.

Twenty years ago or so, the text formatting programs `troff` and `nroff` were developed, as part of UNIX. In these systems, a dot at the start of a line is an escape character, which can be used to call a macro. For example

```
.SH  2.1  "Section heading"
```

might introduce a section.

The author has developed a syntax whose concrete form is similar to the dot syntax of `troff` and `nroff`, but whose abstract syntax is modelled on SGML. This syntax we call the DOT syntax. As in SGML, a tag name can contain digits, period and hyphen as well as letters. As a section is, say a second-level head, one could write

```
.h2 Input syntax
```

to start a section.

In LATEX one might write

```
\documentclass{article}
```

```
\author{Jonathan Fine}
\title{Active \TeX\ and input syntax}
\date{20 January 1999}
```

to start an article. In SGML terms, the author, title and date are all attributes of the article element.

As in SGML, the DOT syntax allows start tags to have attributes. One might write

```
.article  Active &TeX and input syntax
..author  Jonathan Fine
..date    20 January 1999
```

to specify the same information. This double dot notation for attributes is similar to the leading dots notation that TEX the program uses to show the content of boxes [8, page 66]. LATEX does not really have a concept of attributes.

An end tag in the DOT syntax is like so:

```
./article   This is a comment
```

but, as in SGML, end tags can often be implied by the context. For example, if a section cannot contain a section, the start of a new section implies the end of the current one.

SGML has the useful concept of a short reference. In the DOT syntax the start of a line, the end of a line, white space at the start of a line and a blank line are the possible short reference events. One can set matters up so that ordinary lines start paragraphs, blank lines end paragraphs and indented lines commence math mode. Thus the fragment

```
Einstein's famous equation
    E = m c ^ 2
expresses the equivalence of
matter and energy.
```

might be equivalent to

```
Einstein's famous equation
.eq
    E = m c ^ 2
./eq
expresses the equivalence of
matter and energy.
```

but the former is easier both to type and to read.

In summary, the DOT syntax combines the power of SGML with the simple concrete syntax of `troff` and `nroff`. It provides a concrete syntax that ordinary authors can use, whose abstract form is equivalent to that of SGML.

Macro programming

This section is particularly for the TEXnically minded. In Active TEX all characters are active. This is both a problem and an opportunity for the macro programmer. Ordinarily a line in a TEX file such as

```
\def \hello {\message{Hello world!}}
```

would define a macro `hello`, whose execution issues a greeting. This relies on the customary or plain category codes being in force. In Active TEX another approach must be taken.

Ordinarily, control sequences are formed using TEX's eyes. Thus, `\def` in the source file produces the control sequence `def`.

Active TEX uses the mouth of TEX, or more exactly `csname` and `endcsname`, to form control sequences. Macro definitions will be built up using aftergroup accumulation. The plain code line

```
\expandafter \aftergroup
\csname def\endcsname
```

contributes the control sequence `def` aftergroup.

Similarly, the lines

```
\aftergroup {\iffalse}\fi
\iffalse{\fi \aftergroup}
```

contribute left and right braces respectively. Finally, if the macro

```
\def \agchar #1{\expandafter
  \aftergroup \string #1}
```

is passed a character as an argument, it will contribute aftergroup the inert form of this character.

This mechanism allows us to define macros without making use of the ordinary category codes. For example, if we call `begingroup`, then `aftergroup` commands as detailed above, and then `endgroup`, the result could give exactly the same definition of `hello` as at the beginning of this section.

To store such definitions in a file, a syntax is required. Active TEX has been set up so that in a *compiled TEX code* (`ctc`) file, a line such as

```
def hello {message
  {'H'e'l'l'o' 'w'o'r'l'd'!}};
```

has exactly the same effect as the previous definition. Within a `ctc` file, a letter takes itself and all visible characters that follow, and uses `csname`, `endcsname` and `aftergroup` to form and contribute a control sequence. Similarly, active `{` and `}` contribute explicit (or ordinary) begin- and end- group characters `{` and `}` aftergroup. Active right quote `'` is as `agchar` above. Finally, the semicolon `;` closes the existing accumulation group and opens a new one.

This technique of aftergroup accumulation is enormously powerful. It allows arbitrary control sequences and character tokens to be placed into macro definitions. One can even do calculations or pick up values from an external file, as the definitions are being made. Tools are required to make full use of this power.

Suitable content in ctc files allows arbitrary macros to be defined. Active TEX has a development environment, which produces ctc files from suitable source code files. For example,

```
def hello
  { message { "Hello world!" } } ;
```

will when compiled produce the ctc code exhibited above.

Here is another example:

```
def ctc.letter
{
  begingroup ;
    string.visible.chars ;
    let SP endcs ; let TAB SP ;
    let RE suspend.RE ;
    let suspend endcs ;
  xa endgroup xa ag cs ;
}
```

This macro is used within ctc files to produce control sequences aftergroup.

Some comments are in order. Any visible characters can appear in control sequence names. This power should not be abused. We rely on the definitions

```
def string.visible.chars
{
  let lcletter string ;
  let ucletter string ;
  let digit string ;
  let symbol string ;
}
def suspend.RE { suspend ; RE } ;
```

being made already. The tokens SP, TAB and RE in the source file produce (and here it is a mouthful) characters in the ctc file that in turn produce active space, tab and end-of-line characters aftergroup. The tokens xa, ag, cs and endcs in the source file are shorthand for expandafter, aftergroup, csname and endcsname respectively. It is the latter strings that are written to the ctc file by the compiler. Semicolons in macro definitions are for punctuation only. They are ignored. Outside macro definitions they trigger renewal of the aftergroup accumulation group.

This process, of defining macros via ctc files, allows many of the basic problems in TEX macro development to be solved. For example, one can insist that identifiers (tokens in the source file) be declared before they can be used. No more misspelt identifier names! One can also apply a prefix to chosen identifiers, thus segmenting the name space. This will allow a module to control access to its identifiers. No more name clashes!

In the same way, one can use named rather than numbered parameters in macro definitions. For example, instead of

```
\def \agchar #1{\expandafter
  \aftergroup \string #1}
```

as above, one could write

```
def ag.char Char { xa ag str Char } ;
```

where Char has been previously declared to be a macro parameter place holder.

Although this process is somewhat indirect, it does not cause performance to suffer. The compilation process, to produce the ctc files, needs to be done only once, by the macro developer. With modern machines, this does not take long. Similarly, most files will be loaded only once, in the process of making a preloaded format file.

In fact, Active TEX gives two performance benefits. The first is that macro programmers no longer need to resort to tricks, to obtain access to unusual control sequences or character tokens. Thus, more efficient code can be written. The second is that ctc files are generally quite compact. This compression allows them to be retrieved from the hard disc (or network) more rapidly. The DOT syntax gives the same benefits.

Tools for macro programmers are under development. For example, short references will cause indentation to indicate code lines. This section has given examples of what has been done already, and a taste of what lies in the future. The author invites comments.

Processing text

We now turn to the raison d'être of TEX, which is of course typesetting. In §2 we saw how the DOT input syntax allows a document to be broken down into elements with attributes. In §3 we saw some examples of how Active TEX can be programmed. This section is concerned with the content of the document or, more exactly, with the text and the attribute values.

Typesetting plain text, such as

```
This is plain text.
```

is straightforward. Each visible character produces itself, and spaces give interword spaces. Thus the values

```
let lcletter string ;
let ucletter string ;
let digit string ;
let symbol string ;
def SP { unskip ; ~ } ;
```

will, to a first approximation, suffice. (The `unskip` is present so that multiple space characters will count as one. The `~` is Active TEX's way of calling for an ordinary space character.)

Occasionally, the user will want to add *emphasis*. In SGML one uses tags

```
<em>emphasis</em>
```

like so, while in LATEX one uses a macro

```
\emph{emphasis}
```

but in Active TEX one might use

```
Plain text with {emphasis}.
```

for example. Because `{` and `}` are active, they can be programmed to open and close an emphasised group.

This brings us to perhaps the most important concept of this section, which is that of a *data content notation* (DCN). Roughly speaking, such tells us how text is to be processed. For example, the plain text above already has a DCN, namely that it is in English. This is very important if we are using a spell checker or a search engine. Computer programming languages are more formal examples of a data content notation. Mathematics encoded in either plain or LATEX is a third example.

The DOT syntax will be set up so that a data content notation can be associated to the text in each element, and to the text in each attribute value. The DCN will, in a more or less formal manner, tell us what is admissible and how it should be processed. The specification of a DCN is not, however, a matter for the DOT syntax. Rather, it is for the users and experts in the area. Many countries have official bodies that attempt to regulate and bring order to the use, at least in printed form, of a language.

The author suggests as a first step that for plain text a DCN along the following lines be used. For emphasis use `{` and `}`. For bold use `+` and `+`, and for math use `$` and `$`. For verbatim use `|` and `|`. Certain nestings will be prohibited. The following

```
Plain text, {emphasis},
|verbatim| and +bold+,
with some elementary
$2+2=4$ mathematics.
```

is an example of its use.

In math mode, new rules will be required. The author suggests that ordinary TEX but without the backslashes, like so

```
sin ^2 theta + cos ^2 theta = 1
```

as a first approximation. This is only a beginning. Building up a complete system that is capable of handling the complexity of modern mathematical notation, whilst retaining both rigour and ease of use, is not going to be an easy matter. Gaining general acceptance and support of the user community is as much a problem as the formulation and solution of the technical problems.

When SGML is used for markup, there is a tendency to use it for everything, regardless of size. This causes enormous problems to users who either do not have the software tools required, or who prefer to work with plain text files. For example, in the C programming language the `&` operator gives the address of a variable. Code fragments such as

```
ptr = &i ;
```

are common. But in SGML, `&` followed by a letter gives an entity reference, so for an SGML parser to produce the above as output, it must be given

```
ptr = &amp;i ;
```

as input. Something similar happens if one wishes to produce `a<b` as parser output, for the `<b` must not be recognised as a start tag.

Part of the philosophy of the DOT syntax is that it deals with the big things (and also some of the medium sized) while the data content notation deals with the little things. The DOT syntax will have its parser, and each DCN will have its parser. They take turns in processing the input stream.

Let us now return to typesetting. Most of the time, when TEX is typesetting, it is forming either a horizontal list or a math list. Each DCN will, as it processes characters, add items to the current list. Special characters (such as `$`, `{` and `}`) will change the mode in some way. From time to time, say at the end of a title, the current list will be closed and material will be added to the main vertical list, for example. From here on the main difference between plain TEX, LATEX, CONTEXT and Active TEX will be in the libraries of macros used for page makeup, output and so forth.

Typesetting is the purpose of a TEX macro package. TEX was developed to allow typesetting of the highest quality. However, not until the basic functions of Active TEX have been met will it be possible to move on to the typesetting (composition, hyphenation and justification, galleys and page makeup) aspects of the process. Put another way, macro packages such as plain and LATEX have typesetting as their main purpose. Rigour, power and ease of use are the main goals of Active TEX, at least in this stage of its development. A fourth goal is to provide an enduring fixed point for document source files.

History and prospects

Although the basic concept of Active TEX is quite simple — all characters are active — it is surprising just how long it has taken for this idea to emerge. A brief history follows.

In plain TEX the tilde ~ is an active character, which produces an unbreakable interword space, and in math mode apostrophe ' is effectively an active character, used for putting primes on symbols, as in $f'(x)$. Technically, a prime is a superscript. In addition space and end-of-line could be made active, to achieve special results such as verbatim listing of files. In 1987 Knuth [9] wrote about some macros he had written for his wife, in which many of the symbols are active.

In 1990 Knuth froze the development of TEX. In his announcement [10] he wrote:

> Of course I do not claim to have found the best solution to every problem. I simply claim that it is a great advantage to have a fixed point as a building block. Improved macro packages can be added on the input side; improved device drivers can be added on the output side.

In 1992 Fine [2] produced the `\noname` macro development environment, which, like Active TEX, is based on aftergroup accumulation. This solves a major technical problem, namely how to define exactly the macros we wish to define, when the category codes are against us. The `\asts` problem [8, page 373] at the start of Appendix D (Dirty tricks) is solved using aftergroup accumulation.

In 1993 Fine presented a paper [3] to the 1993 AGM of the UK TEX Users Group that contains in embryonic form most if not all of the ideas in this paper. For example, he wrote

> Let us solve all category code problems once and for all by insisting that *the document be read throughout with fixed category codes.*

and then described how '\', for example, could be an active character that parses control sequences, in much the same way as `ctc.letter` does. The paper also contains other prospects that have not been discussed here, such as visual or WYSIWYG TEX (see [7] for a more recent presentation.)

In 1994 Fine [4] argued that the deficiencies in TEX the program had been exaggerated, and that "It would be nice if both TEX and its successor shared at least one syntax for compuscripts that are to be processed into documents" (p. 385). This syntax would have to be rigorous.

In 1994–95 Fine went the whole way, and made all characters active. Using this, he produced a prototype TEX macro package that was able to typeset directly from SGML document files. Due to lack of both sponsorship and commercial interest, the project was left unfinished. This work was presented at the Bridewell meeting (January 1995) on Portable Documents, and published both in Baskerville [5] and MAPS [6], but regrettably not in the special *TUGboat* issue **16** (2) on TEX and SGML, published later that year.

In late 1997 the project was revived, and in May 1998 Fine spoke on Active TEX and input syntax at a meeting of the UK TEX Users Group. Since then a proof-of-concept version of the macros has been available to all those who ask.

There have been other developments that make extensive use of active characters. Michael Downes [1] has done important work on the typographic breaking of equations. He writes (p. 182):

> Some of the changes are radical enough that it would be more natural to do them in LATEX3 than in LATEX2e — e.g., for LATEX3 there is a standing proposal to have nearly all nonalphanumeric characters active by default; having ^ and _ active in this way would have eased some implementation problems.

Werner Lemberg [12] describes the CJK (Chinese, Japanese and Korean) package. This package makes the extended ASCII or eight-bit characters active. He notes (p. 215):

> It's difficult to input Big 5 and SJIS encoding directly into TEX since some of the values used for the encodings' second bytes are reserved for control characters: '{', '}' and '\'. Redefining them breaks a lot of things in LATEX; to avoid this, preprocessors are normally used [...].

Active TEX has been designed from the ground up so that it can go the whole way, and allow problems such as these to be given completely satisfactory solutions, without unnecessary difficulties. The only real price seems to be performance. Because it does much more, it is not as quick as plain TEX or LATEX. This could be a problem for those who use a 286, but on a 486 or better, disk input/output is the real bottleneck.

One of the great things about TEX the program is that it is a fixed point. I would like Active TEX to become a similar fixed point, upon which users can build style files and the like. I would also like the DOT syntax to become a fixed point.

When TEX was developed, Donald Knuth had [8, page vii] the active interest and support of the

American Mathematical Society, the National Science Foundation, the Office of Naval Research, the IBM Corporation, the System Development Foundation, as well as hundreds of more or less ordinary users, many of whom went on to play an active part in the life of the TEX Users Group, and the community generally, and some of whom are still with us.

I firmly believe that Active TEX is a worthwhile idea whose time has come. Please give it your support.

References

[1] Michael Downes. "Breaking equations." *TUGboat* **18**(3), 182–194 (1997).

[2] Jonathan Fine. "The `\noname` macros — A technical report." *TUGboat* **13**(4), 505–509 (1992).

[3] ———. "New perspectives on TEX macros." *Baskerville* **3**(2), 17–19 (1993).

[4] ———. "Documents, compuscripts, programs and macros." *TUGboat* **15**(3), 381–385 (1994).

[5] ———. "Formatting SGML manuscripts." *Baskerville* **5**(2), 4–7 (1995).

[6] ———. "Formatting SGML manuscripts." *MAPS* **14**, 49–52 (1995).

[7] ———. "Editing `.dvi` files, or Visual TEX." *TUGboat* **17**(3), 255–259 (1996).

[8] Donald E. Knuth. *The TEXbook*. Addison-Wesley (1984).

[9] ———. "Macros for Jill." *TUGboat* **8**(3), 309–314 (1987).

[10] ———. "The future of TEX and METAFONT." *TUGboat* **11**(4), 489 (1990).

[11] Sebastian Rahtz. Editorial. *Baskerville* **8**(4/5), 1 (1998).

[12] Werner Lemberg. "The CJK package for LATEX 2_ε: Multilingual support beyond `babel`." *TUGboat* **18**(3), 214–224 (1997).

(untitled)

Oh, what a tangled web is TEX,
what you need to reach success.
To make sure you don't get a reject,
why not get some help from TEX.

—Peggy Kempker

(untitled)

Breathes there the man with the eye so blind
Who never for himself can find
The *cos*sine, 'c' times 'o' times 's'
The `\acronym` run on to the rest
The sentence ending at *et al.*
Although no verb shall yet befall
Until a phrase two lines below
The endquotes where the quotes should go?

If such there be, away go he
To Delaware for a Ph.D.!
The thesis clerk no more shall check;
The only folks to judge his TEX
Are senior faculty, by norm
Concerned with content, not with form
His approval page they'll surely sign;
The publisher, with wit sublime,
His words in XML encases;
And, should he be obscure in places,
With sentence structure ill-prepared,
His authorship will now be shared:

Credit will to the copy editor belong
Grammatically correct, but scientifically wrong.

—Stephen Fulling

Convenient Labelling of Graphics, the WARMreader Way

Wendy McKay
Control and Dynamical Systems
California Institute of Technology
Pasadena, CA 91125
wgm@cds.caltech.edu
http://www.cds.caltech.edu/~wgm/

Ross Moore
Mathematics Department
Macquarie University
Sydney, Australia 2109
ross@mpce.mq.edu.au
http://www.maths.mq.edu.au/~ross/

Abstract

This article describes a system for placing labels on included graphics in a way that does not require the user to be concerned with explicit lengths or coordinates. The full system was developed specifically for use on Macintosh computers but, due to its modularity, can be used with other systems as well.

The WARMreader (read 'Wendy And Ross', selecting either for the 'M') package defines macros to read information from a file, indicating the location of specially marked points where labels may be desired. It also provides a link to the XY-pic macros, which allow arbitrary labels to be attached at these points.

Two applications, *Zephyr* and *Mathematica*, are used to demonstrate techniques for creating files readable for WARMreader, including ways to overcome specific difficulties. Other methods can be used and WARMreader programmed to read the resulting data files.

Various pieces of software and techniques exist for using TEX to put labels onto included graphics. All have significant drawbacks or shortcomings. One method that is widely used, and often recommended as best for Encapsulated PostScript (EPS) files, is to first Typeset the label using *Textures* on a Macintosh, Copy the resulting typeset window, then Paste the clipboard contents into the image file, having been opened within Adobe's *Illustrator* application. Among the drawbacks of this technique are:

- dependence upon a particular computing platform: Macintosh, or PowerMac;
- use of expensive commercial software: Adobe's *Illustrator*, and Blue Sky's *Textures*;
- applicability to just a particular image format: Encapsulated PostScript;
- the original image file must be altered (after copying, please!) to obtain the required results.

Depending upon the working environment, these may not be problems at all; for example, a prepress house would be expected to have the appropriate hardware and software. Similarly an academic may have made the investment to be able to follow this strategy.

However, there is a problem which may cause great difficulties when a manuscript is submitted for publication. Suppose a labelled image needs to be resized or the labels need to be changed for some reason; e.g. the text style chosen does not blend well with the fonts and styles used elsewhere in the publication. Now the EPS file needs to be edited or regenerated in the same way as was done originally. This may no longer be possible — the software used to create it may not be available or the expertise to use it may have been lost.

The WARMreader solution is to use TEX itself, or LATEX, for placing the labels. It uses the XY-pic diagram macros, extending the methods presented at TUG'97 (Moore, 1997), and available on the Web. The idea is to create a coordinate system tailored for the size of the imported image, anchoring labels at appropriate places using these coordinates. This effectively creates an overlay which allows the

labels to seem to be part of the image, when in fact they have been typeset by TEX. WARMreader takes this further, by automating the process so that a user does not have to be concerned with coordinates when specifying the labels. Since the styles and content of the labels are specified within the TEX or LATEX source, there is no need to alter any EPS files. Furthermore, this can be done for graphics of any format that can be included within a TEX document, by whatever means. The only requirement is the ability to create a .bb file,[1] containing information in an appropriate form.

For LATEX, the PSfrag package, as described in *The LATEX Graphics Companion* (Goossens et al., 1997, pp. 460–462), provides similar functionality for EPS files, by treating parts of the file as tags to be later replaced by blocks of TEX-typeset material. This technique has several limitations, apart from being available only for LATEX, and not usable with graphics formats other than PostScript. For best results, the PostScript file "should ideally be designed with PSfrag in mind", and for systematic use, it "requires a good understanding of both the PostScript language and the application generating the figures" (Goossens et al., 1997, p. 462). This is because the replacement portions effectively become part of the PostScript graphic at the point where the tags occur, so are subject to, and must dovetail with, the PostScript graphics state at those places. As this includes color, size, rotation and cropping-region, great care is required to avoid later parts of the graphic obscuring earlier labels or labels being cut off at edges of the graphic. It is not possible to know exactly how the whole thing will appear until the .dvi file has been processed with a PostScript-aware viewer or printer, thus making it tedious to fine-tune the placement of labels.

With WARMreader, the labels can be regarded as occurring within a separate layer, controlled completely from within TEX or LATEX. Any graphic from any source, in any format that can be handled by the TEX installation, can be used as a "backdrop", provided that a suitable .bb file has been prepared. Each of the following three steps can be done quite independently, that is, by different people using different software or techniques:

1. construction of the graphic;
2. make a .bb file, perhaps with text for labels;
3. preparation of code for processing labels within the TEX document.

[1] Such files are used with LATEX's \DeclareGraphicsRule (Goossens et al., 1997, pp. 40–41) for holding just the bounding-box information, since this is all that is needed for TEX to leave sufficient space for an image.

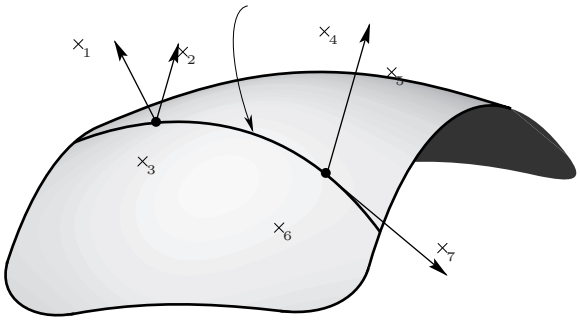


Figure 1: Imported image with "marked points" indicated explicitly.

Only the last *requires* knowledge of TEX or LATEX, though this is desirable if labels are to be completely specified in the .bb file. Indeed it will become apparent below that the greatest control over the final appearance, hence the best results, are obtained when these three tasks are kept completely separate.

Detailed example with marked points

The best way to explain how the WARMreader macros work is with an example. Figure 1 shows an EPS image prepared for a mathematics text (Marsden et al., 2000). Numbered ×s are not part of the image but indicate "marked points", serving as anchors for placement of labels.

Information for the marked points in Fig. 1 is contained in a file named Fig5.4.1.bb, with the graphic itself being named Fig5.4.1.eps. The .bb file gives the natural size (in points) of the imported graphic as well as coordinates for marked points.

In addition to being numbered in sequence, a text string may be given for each marked point. This can be used to help identify why the point has been marked. It may even provide the TEX code intended to be used to specify the label, though it is not at all necessary to use it for this purpose. For example, the code used to produce Fig. 1 was as follows:

```
\begin{xy}
 \xyShowAllMarkedPoints{}{Fig5.4.1}{eps}
\end{xy}
```

Techniques to create a file such as Fig5.4.1.bb (see Fig. 2) are discussed towards the end of this article.

A side effect of \xyShowAllMarkedPoints is to write the coordinates and text strings for each of the marked points into the TEX .log file. The purpose of this is to facilitate preparation of the required labels over several consecutive processing runs.

Adding labels. There are several commands provided for placing labels anchored at marked points.

```
%%Creator: PICT Displayer, by David Rand, Version 1.0, March 1999
%%Title: (Fig5.4.1.eps)
%%Date: 3/13/998h49 PM
%%IMPORTANT: The following BoundingBox indicates only the size of the box, not its position!
%%BoundingBox: 0 0 224 134
%%Coordinates: LL
%%StartMarkedPoints
%%MarkedPoint: ( 38,118)  1 %F_{\lambda}^*t(m_{\lambda})
%%MarkedPoint: ( 77,115)  2 %t(m)
%%MarkedPoint: ( 62, 72)  3 %m
%%MarkedPoint: (130,123)  4 %integral curve of $X$
%%MarkedPoint: (155,107)  5 %t(m_{\lambda})
%%MarkedPoint: (113, 46)  6 %m_{\lambda}=F_{\lambda}(m)
%%MarkedPoint: (174, 38)  7 %X(m_{\lambda})
%%EndMarkedPoints
```

Figure 2: Contents of the file `Fig5.4.1.bb`, containing the "marked point" information for Fig. 1.

```
file: ./Fig5.4.1.bb
Bounding Box is (0,0)->(224,134)
Marked '1' point at ( 38,118) for F_{\lambda }^*t(m_{\lambda }).
Marked '2' point at ( 77,115) for t(m).
Marked '3' point at ( 62, 72) for m.
Marked '4' point at (130,123) for integral curve of $X$.
Marked '5' point at (155,107) for t(m_{\lambda }).
Marked '6' point at (113, 46) for m_{\lambda }=F_{\lambda }(m).
Marked '7' point at (174, 38) for X(m_{\lambda }).
Found 7 data points.
```

Figure 3: Marked point information, as it appears in the `.log` file.

The simplest, but not always the most effective, of these is useful when the required labels are provided as the text string accompanying each marked point. For the moment, ignore the ⟨*mods*⟩ parameter; it will be explained later.

`\xyMarkedTxt` ⟨*mods*⟩{⟨*num*⟩}
`\xyMarkedText` ⟨*mods*⟩{⟨*num*⟩}
`\xyMarkedMath` ⟨*mods*⟩{⟨*num*⟩}
`\xyMarkedTxtPoints` ⟨*mods*⟩{⟨*list*⟩}
`\xyMarkedTextPoints` ⟨*mods*⟩{⟨*list*⟩}
`\xyMarkedMathPoints` ⟨*mods*⟩{⟨*list*⟩}

The first two commands are just alternative names which give identical results. These, and the third command, set the supplied text-string as a label at marked point number ⟨*num*⟩, assuming it to contain TEX code valid in text or math mode, as the name suggests. Several marked points are handled simultaneously by the remaining commands, where the ⟨*list*⟩ consists of numbers and number ranges. Note that the fourth and fifth commands are simply alternative names which give identical results. With `\xyMarkedMathPoints`, the strings in the `.bb` file are presumed to be valid math-mode source, *without* the need for surrounding `$...$` delimiters.

In Fig. 3 it can be seen that point number 4 requires text mode whereas all others are meant for math mode. One way to do this is with the following code, which yields the results in Fig. 4:

```
\WARMprocessEPS{Fig5.4.1}{eps}{bb}
\renewcommand{\labeltextmodifiers}{++!D}
\renewcommand{\labelmathmodifiers}{+!D}
\renewcommand{\labelmathstyle}{\scriptstyle}
\renewcommand{\labeltextstyle}{\footnotesize}
\begin{xy}
 \xyMarkedImport{}
 \xyMarkedMathPoints{1-3,5-}
 \xyMarkedTextPoints{4}
\end{xy}
```

Note the following points:

- The `\WARMprocessEPS` command uses its arguments to specify the graphic image and the file to read for the marked-point information.
- The expansion of `\labeltextmodifiers` yields Xy-pic ⟨*modifier*⟩s that affect the way a label is positioned with respect to its marked point, when using `\xyMarkedTextPoints` and other text mode commands. For math-mode labels there is `\labelmathmodifiers`.

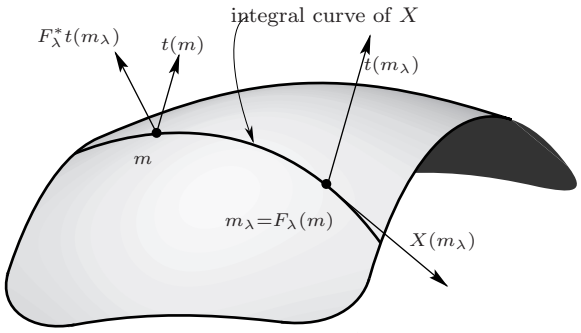


Figure 4: Imported image, with attached labels.

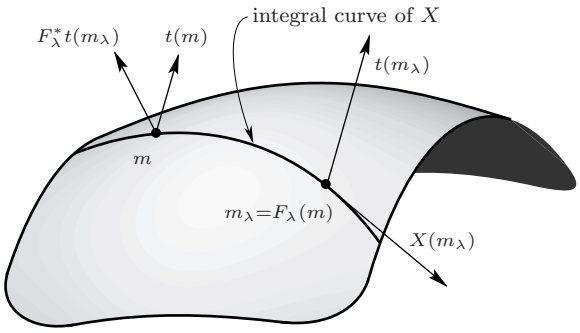


Figure 5: Imported image, with fine adjustments to the positions of labels.

- `\xyMarkedImport` extends the XY-pic command `\xyimport`. Its argument can be the name of the graphics file to be placed into the TEX or LATEX document. However, it is not required when `\WARMprocessEPS` has been used already.
- A ⟨*list*⟩ can be a comma-separated list of numbers or numeric ranges, *a*-*b*.

For extra convenience in specifying lists, the following commands are also available to put labels on all but a specified ⟨*list*⟩ of marked points:

```
\xyMarkedTxtExcept ⟨mods⟩{⟨list⟩}
\xyMarkedTextExcept ⟨mods⟩{⟨list⟩}
\xyMarkedMathExcept ⟨mods⟩{⟨list⟩}
```

The empty ⟨*list*⟩ always means to use *all* marked-points, regardless of the '`Except`'. Also, open-ended ranges such as `-3` and `5-` refer to all numbers to or from the appropriate extremity.

Commands for styled labels. As well as commands listed above, font size and style for text and math labels can be specified, using commands:

```
\xyMarkedStyledTxt ⟨mods⟩{⟨style⟩}{⟨num⟩}
\xyMarkedStyledText ⟨mods⟩{⟨style⟩}{⟨num⟩}
\xyMarkedStyledMath ⟨mods⟩{⟨style⟩}{⟨num⟩}
\xyMarkedStyledTxtPoints ⟨mods⟩{⟨style⟩}{⟨list⟩}
\xyMarkedStyledTextPoints⟨mods⟩{⟨style⟩}{⟨list⟩}
\xyMarkedStyledMathPoints⟨mods⟩{⟨style⟩}{⟨list⟩}
\xyMarkedStyledTxtExcept ⟨mods⟩{⟨style⟩}{⟨list⟩}
\xyMarkedStyledTextExcept⟨mods⟩{⟨style⟩}{⟨list⟩}
\xyMarkedStyledMathExcept⟨mods⟩{⟨style⟩}{⟨list⟩}
```

Allowable values for ⟨*style*⟩ in text mode are macro names that can sensibly be used with XY-pic's `\txt` command:

```
*⟨modifiers⟩\txt⟨style⟩{...balanced text...}
```

while for math mode ⟨*style*⟩ must work within in-line mathematics as follows:

```
$⟨style⟩{...balanced math...}$ .
```

Fine adjustment of labels. The labelled image in Fig. 4 looks quite good but there are blemishes: e.g. the text label "integral curve..." overlaps with the curved arrow, the math label "$m_\lambda = \ldots$ " is too far from the large dot which it is meant to be labelling, and the "$t(m)$" and "$t(m_\lambda)$" are perhaps too close to the arrows they are meant to label.

The position of the text label, at marked-point number 4, could be adjusted by choosing a different set of XY-pic modifiers for the expansion of the macro `\labeltextmodifiers`. This works when there is just a single label to fine-tune but is no good when more than one needs special adjustment.

To allow many specialised adjustments, all the commands introduced so far allow XY-pic modifiers to be specified. These come immediately after the command-name, but *before* the opening brace:

```
\xyMarkedMathExcept ⟨mods⟩{⟨list⟩}
\xyMarkedStyledPoints ⟨mods⟩{⟨style⟩}{⟨list⟩}
\xyMarkedStyledTxtPoints ⟨mods⟩{⟨style⟩}{⟨list⟩}
```

The ⟨*mods*⟩ are just XY-pic ⟨*modifiers*⟩, here given a shortened name to fit the column width.

Figure 5 shows how this could be done. The source code is as follows. Note how three of the math labels are positioned with explicit XY-pic modifiers while the others use `\labelmathmodifiers`. The single text label is also positioned explicitly, to good effect, so there is no need for `\labeltextmodifiers`.

```
\WARMprocessEPS{\exnamei}{eps}{bb}
\renewcommand{\labelmathmodifiers}{+!D}
\renewcommand{\labelmathstyle}{\scriptstyle}
\renewcommand{\labeltextstyle}{\footnotesize}
\begin{xy}
 \xyMarkedImport{}
 \xyMarkedMathPoints ++!D!L(.1){2}
 \xyMarkedMathPoints +!D!L(.3){5}
 \xyMarkedMathPoints ++!D{6}
 \xyMarkedMathExcept{2,4-6}
 \xyMarkedTxtPoints ++!D!L(.2){4}
\end{xy}
```

Recall the effect of the XY-pic modifiers, e.g. `+!D!L(.3)`. First, TEX sets an `\hbox` containing the typeset label. Usually this box is centered, so that if there were no ⟨*modifier*⟩s the center of the label would be anchored at the marked point. The modifier `+` adds a small margin, increasing the size of the box both vertically and horizontally. Next the `!D` shifts the reference point (Down) within the box to the bottom edge; with no further modifiers, the label now appears entirely *above* the position of the marked point, with the bottom edge occuring the width of the margin away from it. Finally the `!L(.3)` nudges the reference point towards the Lefthand edge, by an amount .3 of the distance to it, so that now more of the label appears on the righthand side of the marked point.

Note that nudging using `!D`, `!L`, `!R` and `!U` (Up), has the effect of shifting the label in the *opposite* direction to the specified nudge. Numerical ⟨*factor*⟩s, such as `(.3)`, are optional; if omitted, the reference point is moved all the way to the specified edge. [2]

Strategies for marking points. Figure 5 shows how labels can be accurately positioned, using the locations of the marked points of Fig. 1. The marked points are away from "busy" parts of the graphic. They indicate where labels can be placed near to that part of the image being labelled yet not interfere unduly with other parts of the image.

While this is an intuitive strategy for selecting places to be marked, it can mean that adjustments, by "nudging", are required to position the labels to best effect. Some trial-and-error is usually required before finalising the positions of all labels by choosing the best ⟨*factor*⟩s.

Marking the busy places. In many cases it is a better strategy to put marked anchor points much closer to the places to which the labels *refer*, rather than to where the labels themselves are desired. In Fig. 6 we see the same image as previously but with a different set of marked points for the same labels. For this set it is sufficient to use just a new file (`Fig5.4.1.bb2`) for the labels while retaining the same file (`Fig5.4.1.eps`) for the image itself. Indeed, that image is used 6 times in this paper, yet only one copy of the file is required.

```
\WARMprocessEPS{Fig5.4.1}{eps}{bb2}
\renewcommand{\labelmathstyle}{\scriptstyle}
\renewcommand{\labeltextstyle}{\footnotesize}
\begin{xy}
 \xyMarkedImport{}
 \xyShowMarkPoints{*++[red][F-:red]@{*}}{-}
 \xyMarkedMath +!DR{1}
 \xyMarkedMath +++!D{2}
 \xyMarkedMath +!U{3}
 \xyMarkedTxt +!DL{4}
 \xyMarkedMath +!L{5}
 \xyMarkedMath +!UR{6}
 \xyMarkedMath +!DL{7}
\end{xy}
```

[2] Refer to the XY-pic Reference Manual (Rose and Moore, 1999), for details of the XY-pic language for structured diagrams.

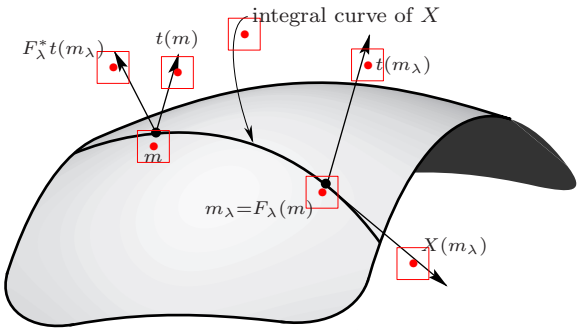


Figure 6: Attaching labels by corners and edges to places near to what they refer.

When the marked points are chosen this way, the labels can usually be well positioned by specifying just margins and an edge or corner to be where the reference point of the label should occur. There is little need for delicate nudging and ⟨*factor*⟩s.

On the other hand, extreme accuracy is not at all necessary when choosing positions for the marked points. In this article, the `.bb` files were generated using low-resolution preview images. These need *not* be accurate scaled-down versions of the higher resolution images rendered by PostScript. Inaccuracies can be compensated for using XY-pic adjustments.

Adjusting sizes and styles. Another significant advantage of this strategy becomes apparent when the image or labels need to be resized or restyled, perhaps for use in a different context. This will almost certainly change the relative size of the labels and the image. Smaller-sized labels remain anchored to places near to what they refer. On the other hand, relatively larger labels can have been anchored so as to expand over portions of the image that are otherwise empty. In either case there may be no need to make any adjustments to the coding of labels.

```
\WARMprocessEPS{Fig5.4.1}{eps}{bb2}
\renewcommand{\labelmathstyle}{\displaystyle}
\renewcommand{\labeltextstyle}
  {\large\bfseries\sffamily}
\begin{xy}
 \xyMarkedImport{}
 ...
 ...
\end{xy}
```

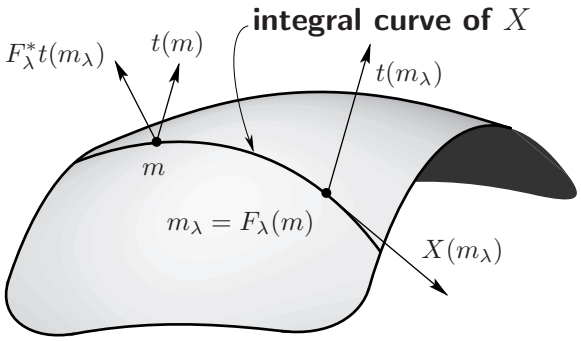


Figure 7: Labels remain well positioned with relative changes of scale.

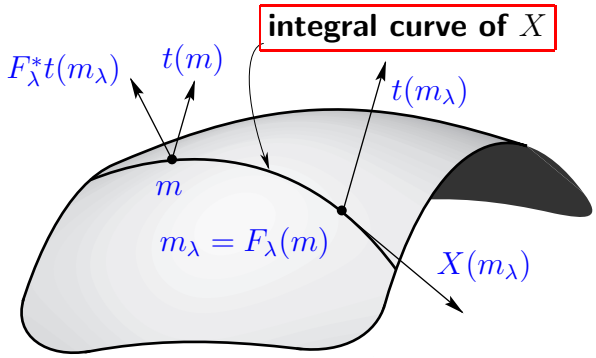


Figure 8: All labels dropped as XY-pic styled text boxes.

To LaTeX or not to LaTeX. Although the above examples have used LaTeX, the WARMreader macros work equally well with plain TeX, and most other formats, as does XY-pic. The only requirement is to be able to import the graphic and customise the expansion of a single macro, \xyWARMinclude, to suit. This macro takes as argument the name of the image file. As a practical default, it expects to be able to use the \includegraphics command from LaTeX's graphics package:

```
\def\xyWARMinclude#1{\includegraphics{#1}}
```

This definition can be overridden by replacing the \includegraphics with \psfig or \epsfig or \epsfbox or other command for placing an imported graphic within the TeX or LaTeX document.

There must be only one argument for the filename. The result should be an \hbox of exactly the size required for the image to occupy. (This is so that \xyWARMinclude{⟨*filename*⟩} can be used as the argument to an \xyimport command.)

Note that some macros for including graphics are *not* suitable. For example, the \centerpicture macro from *Textures*' picmacs.sty file cannot be used since it inserts stretchable 'glue' to span the whole page width; on the other hand, \picture from the same file can be used.

Rotations and scaling. The requirements stated in the previous subsection allow scaling, rotating and resizing of imported graphics. For example, a rescaling can be achieved using LaTeX as follows:

```
\newcommand{\scaledfig}[2]
  {\scalebox{#1}{\includegraphics{#2}}}
\renewcommand{\xyWARMinclude}[1]
  {\scaledfig{.7}{#1}}
```

It is only the image which is resized or rescaled; the size and style of labels is controlled independently, as discussed above. When different images require different scale factors, then the definition of \scaledfig belongs in the document preamble and a re-definition of \xyWARMinclude should precede a figure, if needed. (See Fig. 13 for an example.) Optional arguments to \includegraphics or other command can be incorporated in a similar way.

Same locations, different labels. It is not necessary to use the text strings from the .bb file for the labels. be specified within the TeX or LaTeX source. This is most convenient, since it means that:

- changes can be made to the labels without the need to make any adjustments to the .eps or .bb files;
- the same image can be used many times with different labels;
- labels may cross-reference other parts of the document; in a web document the labels could become hyperlinks, as in an "image-map".

Figure 8 uses this technique as one way to get larger sized mathematics in labels. The actual code used is shown in Fig. 9.

Using \xyMarkedPos allows the most flexibility amongst all the commands available for placing a label. Essentially all that it does is to move the XY-pic "current point" to the location of the marked point. Now any valid XY-pic code can be used to place anything at all at that point.

Commands to allow direct use of XY-pic code at the marked points are as follows:

```
\xyMarkedPos{⟨num⟩}⟨pos⟩*⟨object⟩
\xyShowMark{⟨pos⟩*⟨object⟩}{⟨num⟩}
\xyShowMarkPoints{⟨pos⟩*⟨object⟩}{⟨list⟩}
\xyShowMarksExcept{⟨pos⟩*⟨object⟩}{⟨list⟩}
```

In the latter three cases, if the {⟨*pos*⟩*⟨*object*⟩} is empty, then a default \markobject is used for each point in the ⟨*list*⟩. This is the same for the command

```
\WARMprocessEPS{\exnamei}{eps}{bb2}
\renewcommand{\labeltextstyle}{\large\bfseries\sffamily}
\begin{xy}
 \xyMarkedImport{}
 \xyMarkedPos{1}*+!DR[blue]\txt\labeltextstyle{$F_{\lambda}^*t(m_{\lambda})$}
 \xyMarkedPos{2}*+++!D[blue]\txt\labeltextstyle{$t(m)$}
 \xyMarkedPos{3}*+!U[blue]\txt\labeltextstyle{$m$}
 \xyMarkedPos{4}+/u1ex/*+!DL[F-:red]\txt\labeltextstyle{integral curve of $X$}
 \xyMarkedPos{5}*+!L[blue]\txt\labeltextstyle{$t(m_{\lambda})$}
 \xyMarkedPos{6}*+!UR[blue]\txt\labeltextstyle{$m_{\lambda}=F_{\lambda}(m)$}
 \xyMarkedPos{7}*+!DL[blue]\txt\labeltextstyle{$X(m_{\lambda})$}
\end{xy}
```

Figure 9: Coding for Fig. 8 uses various XY-pic effects.

`\xyShowAllMarkedPoints`, as was used in Fig. 1 and Fig. 6. All these commands finish with the XY-pic `\POS`-parser command so that further XY-pic drawing can be done, if desired. For a single marked point located using `\xyShowMark`, its number is also placed, using a macro `\markobjectlabel`. This expands as follows; it can be redefined if desired.

```
\def\markobjectlabel#1{\POS*\dir{x},
  *+<3pt>!U{\scriptscriptstyle#1}}
```

Symbolic names. Although all the examples so far have referred to the marked points by number, they can instead be assigned a symbolic name. Any text string suffices instead of the number within the `.bb` file. This string can be used instead of the ⟨*num*⟩ in those macros that require such an argument. Macros wanting a ⟨*list*⟩ still work since there is an internal counter as well as the symbolic name.

Format of the `.bb` files. The examples shown here have used `.bb` files in which the information is presented as in Fig. 2. This form is based on the structure of comments in PostScript files. Note how it includes a `%%BoundingBox` comment in the standard PostScript form as well as the actual marked point information.

Indeed, it is the presence of this comment that warrants the use of the extension `.bb`. In LaTeX, the `\includegraphics` command can make use of the bounding-box information contained in a file with this extension. For an EPS graphic this information could be read from the `.eps` file itself; however, since these files can be very large and can contain binary portions which TeX does not handle easily, it is often more convenient to have it extracted into a separate `.bb` file. For non-EPS graphics, all TeX requires is the bounding-box information to know how large an empty box to leave while typesetting. Having this in a separate `.bb` file is the only viable option due to the binary nature of most graphics formats. With `WARMreader`, this use of a `.bb` file has been extended to include extra marked point information.

Furthermore, with a `.eps` or other PostScript image, the contents of a `.bb` file can be pasted into the `.eps` file for easier distribution. When there is initially no `.bb` file, the `WARMreader` macros search the `.eps` file instead and a `.bb` file is created, containing the `%%BoundingBox` comment. The marked point information is included also, provided that a `%%StartMarkedPoints` comment has been encountered within the first 20 lines.

It is now apparent that the labelling strategy discussed here can be used with any graphics format provided that:

- the TeX installation has a way to specify that the image file is required within the `.dvi` or other output format being produced;
- a file is available, containing the size and all the marked point information, using numbered or symbolic names and (optionally) text strings.

Making `.bb` files with *Zephyr*

With a Macintosh system, the easiest way to create `.bb` files for EPS graphics, and other formats, is to use David Rand's *Zephyr* (1999) text and list editing program. After launching the application, a graphics file is opened by selecting the special PICT Displayer extension from the pull-down menu, as shown in Fig. 10. Choosing Display LL Coordinates prepares *Zephyr* for recording coordinate values for marked points, where the origin is at the lower left of the image. This opens a file-dialog window, allowing the required file to be found and opened. Indeed, any file that contains a graphics preview image, in the Macintosh PICT format, can be selected from the file-dialog. It is this preview image which will be shown and used for marking points.

The Display UL Coordinates alternative can be chosen instead, to have the origin at the upper-left

Figure 10: Opening a graphics file with the PICT Displayer in *Zephyr*.

corner and with the second coordinate increasing downwards. If this is done, images in the TEX document using such coordinates should be preceded by the `\MacintoshOrigin` command.

To mark a point within the image, simply click with the mouse at the desired point. A small window will pop up, as in Fig. 11, allowing a label to be typed and the selection confirmed.

After the first point has been chosen a Log-window appears, containing bounding-box and other information, as well as data for the first marked point. A line of data is added for each subsequent point. Within the image the point is marked by a numbered cross. Guide-rules help position the cursor accurately: gradations may be inches, centimeters, or pixels. The Log-window can be seen in Fig. 11. Since it contains just plain text, the Log can be edited at any time.

When finished with an image, click its close-box (in the upper-left corner); this also adds the closing comment to the Log. Finally close the Log-window and Save As..., choosing whatever name is desired — usually ending `.bb`, though this is not compulsory.

`\MacintoshOrigin`	allow for coordinates with origin at upper-left
`\EndLineAdjust`	adjust for awkward line-end characters

End-of-line problems. Text files created on one computing platform do not always transfer to other platforms in a way that allows them to work correctly. This can happen with `.bb` files. Declaring `\EndLineAdjust` before processing the `.bb` file may alleviate a TEX error that otherwise can occur.

Annotations on *Mathematica* graphics

The next examples have been used for teaching elementary mathematics. They were constructed using the *Mathematica* (Wolfram, 1994) software package and saved in EPS format. In fact there is more

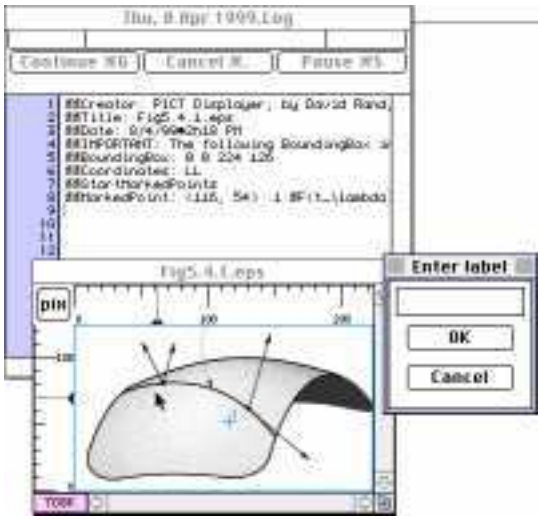


Figure 11: Marking points for the `.bb` file with *Zephyr*'s PICT Displayer.

than one way to do this with *Mathematica*, which can produce `.eps` files having quite different structure and properties. The first example is in direct analogy with techniques discussed already. Extra considerations apply when the `.eps` file contains an `%%AspectRatio` comment, as in later examples.

The *Mathematica* Front-End software allows for "point-&-click" on images to obtain coordinates,[3] in the coordinate system used to calculate the image contents. This technique was used to create data files for the remaining examples; an extension `.mbb` indicates their origin.

```
\WARMprocessMMA{Q1}{eps}{mbb}
\renewcommand{\xyWARMinclude}[1]
  {\scaledfig{.7}{#1}}%
\begin{xy}
\xyMarkedImport{}
\xyMarkedPos{para}*+{}
  ,\ar@{<-}+(.7,25)*+!D\txt{base of parabola}
\xyMarkedPos{cub1}*++{}!D(.6),\ar@{<-}-(3.5,15)
 *+!U(.8)\txt{turning points\\of a cubic}="cub"
\xyMarkedPos{cub2}*++{},\ar@{<-}"cub",
\xyMarkedPos{neg1}*++{},\ar@{<-}+(.5,-25),
 *+!L(.6)\txt{local minima}="min"
\xyMarkedPos{neg2}*++{},\ar@{<-}"min",
\end{xy}
```

Click at the four edges to get the bounding-box information. Some manual editing is needed to put this into the form shown in Fig. 12. The TEX source uses the macro `\WARMprocessEPS` to read size and

[3] First click once on an image to select it, then hold down the modifier-key while clicking at the desired places within the image. When done, choose the Copy menu-item. Subsequently Paste the contents into an editable cell.

```
LDRU:{-3.89059, -43.2333, 4.21704, 43.0523}
StartData
,{1.5145, 5.44064, para}
,{2.03422, -9.67778, cub1}
,{-1.01481, 17.6091, cub2}
,{0.96013, 2.49071, neg1}
,{-1.04945, 2.49071, neg2}
EndData
```

Figure 12: Listing of `Q1.mbb`, containing the marked-point data for Fig. 13.

marked-point data. Having just a symbolic label for each point is quite sufficient for Fig. 13, in which the marked points are not where labels occur but are near the endpoints of arrows. Positions for the labels are determined relative to these arrow-ends, using XY-pic commands. Notice how some labels are positioned relative to one marked point, then used to draw an arrow to another.

Adjusting for aspect ratio. Some graphics export options in *Mathematica* result in graphics for which the bounding-box is not the same size or shape as the preview image. For instance, some have a rectangular preview but `%%BoundingBox` for a square enclosing the image.

```
%%AspectRatio: .61803
%LDRU:{-2.42465, -3.80861, 6.6868, 3.25092}
LDRU:{-2.18, -3.57, 6.50, 3.24}
StartData
,{2.29262, 2.27719, 3sinX}
,{1.54949, -0.927998, sin3X}
EndData
```

The "aspect ratio" (i.e. height/width) of the rectangle must be known to handle such graphics correctly with `WARMreader`. This can be obtained from the `.eps` file, where it is given as a PostScript-like comment; it must be supplied as the first line in the `.mbb` file. The `\WARMprocessMMA` macro is replaced with a variant called `\WARMprocessMMAR`.

Such images sit badly in a TEX document without removing the extra space below, when the aspect ratio is greater than 1, or at left and right, when the aspect ratio is less than 1. This explains the `\vskip` commands in the following listing for Fig. 14.

```
\WARMprocessMMAR{QA1}{eps}{mbb}%
\renewcommand{\xyWARMinclude}[1]
  {\scaledfig{.7}{#1}}%
\newcommand{\Xhair}{%
 \drop[thinner][red]+[o][F-]@{x}}%
\vskip-3.25\bigskipamount
\begin{xy}
 \xyMarkedImport{}
,(0,0)\Xhair,(0,3)\Xhair,(0,-3)\Xhair
,(6.2831,0)\Xhair,(-1.5708,1)\Xhair
```

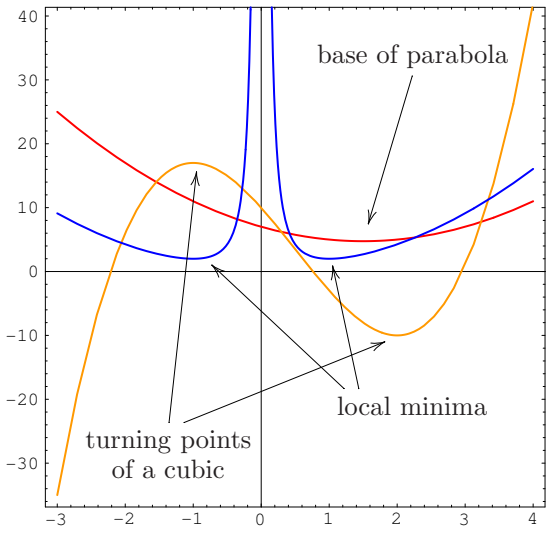


Figure 13: Labelled graphic, using *Mathematica* and `WARMreader`.

```
,(-1.5708,-3)\Xhair,(1.5708,3)\Xhair
,(.5236,1)\Xhair,(3.6652,-1)\Xhair
 \xyMarkedPos{3sinX},*++!L{3\sin x}
 \xyMarkedPos{sin3X}*+{}
  ,\ar@{<-}+(.7,-1)*+!U!L(.4){\sin 3x}
\end{xy}%
\vskip-3.5\bigskipamount
```

In most cases this is enough for good placement of labels over the imported image; fine-tuning can be done using XY-pic modifiers, as described earlier. If greater accuracy is required in establishing the coordinate system over the image, some tweaking of the bounding-box may be done inside the `.mbb` file, as in the third line of the above listing for Fig. 14. The second line, which shows the coordinates obtained from edges of the preview image, has been suppressed to allow the following line to give modified values. Note how the cross-hairs have been accurately positioned.

A further complication occurs when the graphic contains wide axis labels or tick marks. Now not all edges of the preview image need correspond to edges of the bounding box, when printed on the page. *Mathematica* rescaled the preview to include the axis labels but, on the printed page, the main part of the image is larger, with the axis labels extending into the extra space due to the aspect ratio.

To get best positioning, some visual estimation is required. An extra offset parameter is supplied with the `%%AspectRatio` comment, to measure the extent that labels would fall outside the bounding-box, if it had been rectangular, not square.

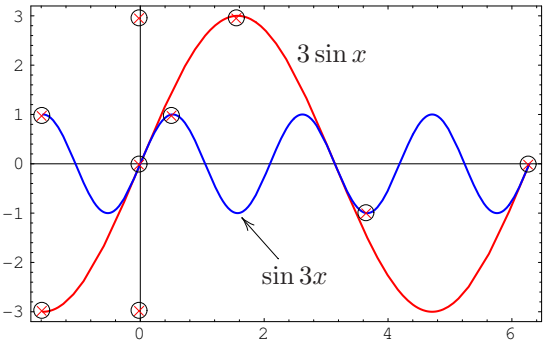


Figure 14: *Mathematica* graphic having aspect ratio $\neq 1$. Cross-hairs superimposed at fractional multiples of π, indicate accuracy of the alignment.

```
%%AspectRatio: 1.6 :1.294
%LDRU:{-5.24417, -171.787, 8.21328, 248.046}
LDRU:{-3.95, -148, 8.21328, 226}
StartData
,{-1.13851, 16.7698, amax}
,{4.94396, -92.2394, amin}
,{2.05478, 0.565704, bflat}
EndData
```

In the above listing of the `.mbb` file for Fig. 15, the third line gives the extents of a rectangle, with aspect ratio 1.6, that just encloses the height of the graphic. The left-hand edge of this rectangle falls roughly $1.294 = 5.24417 - 3.95$ horizontal units from the edge of the axis labels on the left.

Other formats for `.bb` data

The `WARMreader` macros can be used to read data for marked-points from files having other formats. For a given format one needs to specify 'data-start' and 'data-end' strings, as well as patterns to be used with macros to extract the necessary components of the bounding-box and the lines of marked-point data. Stripped-down versions of these patterns are also needed, to help determine when a line does *not* match what is required. Also required is a token list, to hold the expansion part of a TEX macro to interpret the data which matches the supplied pattern. This macro must store the data appropriately for later use. Finally, there must be a TEX macro that controls the order in which the various steps are performed; i.e. reading the data file with the appropriate pattern to interpret each data line. For more specific information on what is required, consult the file `WARMreader.sty`.[4]

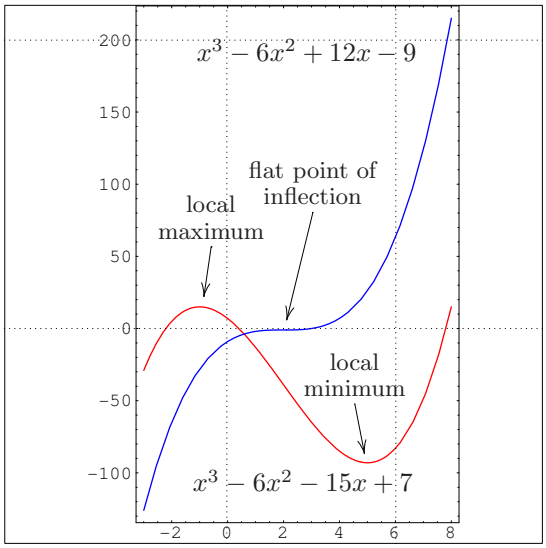


Figure 15: *Mathematica* graphic with non-trivial aspect ratio and relatively wide axis labels. The frame shows the oversized bounding-box, while dotted grid-lines indicate the accuracy of the alignment.

References

Goossens, Michel, S. Rahtz, and F. Mittelbach. *The LaTeX Graphics Companion*. Addison-Wesley, 1997.

Marsden, Jerrold E., R. Abraham, and T. Ratiu. *Manifolds, Tensor Analysis, and Applications*. Springer-Verlag, 2000. (in prep.; 2nd ed. 1988).

Moore, Ross. "High quality labels on included graphics, using XY-pic". *TUGboat* **18**(3), 159–165, 1997. On-line version at `http://www-texdev.mpce.mq.edu.au/XyPIC/XYarticle/`.

Moore, Ross. "Erratum: High quality labels on included graphics, using XY-pic". *TUGboat* **19**(1), 61, 1998.

Rand, David. "Zephyr, A list and text editor for the Macintosh". Shareware software available online at `http://www.crm.umontreal.ca/~rand/Zephyr_Eng.html`, 1999.

Rose, Kristoffer H. and R. R. Moore. *XY-pic Reference Manual, version 3.7*. DIKU, University of Copenhagen, 1999.

Wolfram, Stephen. *Mathematica, A System for Doing Mathematics by Computer*. Addison-Wesley, 2nd edition, 1994. `http://www.wri.com/`.

[4] Available from `http://www-texdev.mpce.mq.edu.au/TUG/WARM/WARMreader.sty` .

Sergey Lesenko and Laurent Siebenmann

Viewing DVI files with Acrobat Reader: DVIPDF gives birth to AcroDVI

Sergey Lesenko
Institute for High Energy Physics (IHEP)
Protvino (Moscow Region)
142284 Russia
lesenko@mx.ihep.su

Laurent Siebenmann
Mathématique, Bât. 425
Université de Paris-Sud
91405-Orsay, France
Laurent.Siebenmann@math.u-psud.fr

Abstract

The first author's DVIPDF program converts from DVI, the output format of TEX, to PDF, the input format for Adobe's Acrobat Reader. Although DVIPDF has existed as a prototype for about three years, the uses to which it will be put by the TEX community are only gradually emerging. This article presents one concrete application. DVIPDF has been adapted under the Windows 9X/NT operating systems to allow "drag-and-drop" viewing of DVI files in Acrobat Reader. The resulting viewer is called AcroDVI: it involves DVIPDF and the Acrobat Reader, operating in concert. Intended for viewing legacy DVI files, it aims to support the most common `\special` commands. An evolutive change in electronic publishing practice is proposed in this connection: the conventional EPS graphics format could well be replaced by various optimal formats: JPEG or PNG for bitmaps, and PDF for vectorial graphics. These can then be conveniently exploited in some natural ways hitherto unavailable: shared, re-edited, or directly viewed.

Introductory viewing experience

> *... Egli e' scritto in lingua matematica, e i caratteri son triangoli, cerchi, ed altre figure geometriche ...*
>
> — Galileo, writing on physical science

To view an article, the modern scientist using AcroDVI can simply push its DVI file icon onto the icon of AcroDVI. The DVI file is quickly converted to PDF; then a window pops up for viewing by Acrobat Reader. If you are not already familiar with Acrobat Reader, the biggest thrill will surely be the top-quality graphics and typography, both superior in various respects to what web browsers offer. Worth noting for TEX users are the hypertext features familiar from web browsers. All this is remarkable, but only the notion that DVI can be the root format is new.

In the AcroDVI viewing experience, even those familiar with Acrobat Reader will enjoy one novelty: enhanced visibility of the graphics objects. They appear not only in the PDF page view but also autonomously in native formats suitable for reuse and also for display at an optimal scale. The three formats that AcroDVI deals with directly are PNG (Portable Network Graphics) for bitmapped high contrast graphics, JPEG (Joint Photographic Experts Group) for color photos, and PDF (Portable Document Format) for vectorial graphics (more on these later). One of these formats should be optimal for just about any still (that is, non-moving) graphics object.

If you push the icon of a PNG or JPEG or PDF file onto that of AcroDVI, then it will be immediately viewed in an Acrobat Reader window. Likewise for EPS files, provided Acrobat Distiller or *Ghostscript* is accessible and enough fonts are available. Latent in Acrobat Reader, which does not directly process PNG or JPEG files, are broad graphics viewing capabilities, and DVIPDF has

merely tapped into them; Adobe could have done as much for Acrobat Reader, but chose not to.

There are many specialized tools for both viewing *and* editing PNG and JPEG files, notably the free XNview under Windows and Linux and the shareware program Graphics Converter on the Macintosh. If you take care to view at scale 100%, then you will see the bitmapped graphics at their best possible quality.

The most widely used tools for viewing PNG and JPEG graphics are probably the web browsers. This is an open invitation to make double use of the graphics in an article: first, in an illustrated HTML introduction, and second, in the DVI file for the article's body. Thus, AcroDVI provides polyvalence for graphics. At the same time, it provides a basic polyvalence for text, namely, the possibility to view the same DVI file with Acrobat Reader and with traditional DVI viewers.

The need for polyvalence and low bulk was the immediate motivation for developing AcroDVI. It arose for mathematics journal content in the CD-ROM project called MathCD, for which the second author is managing editor. Indeed, MathCD has an order of magnitude less space available for many journals than a single journal can afford to use on the Internet.

Where space is at a premium, as on some CD-ROMs and in personal electronic libraries, the DVI format plus auxiliary native graphics can now reasonably replace the PDF format. On the other hand, where space is virtually unlimited, as on many Internet sites, expect to see more formats and greater bulk.

There are relatively few hyper-references in the electronic journal articles on MathCD. Currently, DVIPDF does support hyper-references using a `\special` syntax, parallel to Acrobat Distiller's `pdfmark` syntax. However, it does not yet support the most common `\special` syntax of today's DVI files, namely the one introduced by *xhdvi* and paralleling HTML.

What is AcroDVI really?

The technologically aware user will tend to see AcroDVI as the sum of its parts: DVIPDF plus Acrobat Reader plus some Windows programming using the Dynamic Data Exchange (DDE) protocol as the framework for collaboration between DVIPDF and Acrobat Reader.

However, to the passing user, the whole will be more important than the parts. That is, AcroDVI acts as a viewer that directly accepts most DVI files (*modulo* font availability), as well as graphics files in the PNG, JPEG, or PDF formats — it is the first viewer to do all this.

We have decided to dignify the whole with the the acronym AcroDVI. A viewing eye is what you should see in the logo:

AcroDVI

that is put together by a TEX macro `\AcroDVI` from pieces of standard TEX fonts.

The Windows icon is a colored iris (an "eye-con"); files to be viewed are dragged and dropped on top of the icon. (See Fig. 1 for black-and-white renditions of the current icon.)

In its present provisional state, AcroDVI involves a single binary executable called `dvipdf.exe` while the shortcut icon to it and the distribution directory are called AcroDVI. This makes DVIPDF and AcroDVI rather like a marsupial 'mother-with-baby-in-pouch'.

To facilitate portability, source code is divided into modules of C++ source code devoted exclusively to the AcroDVI viewer functions and modules of C code that can hopefully be compiled as a "black box" processor to implement DVIPDF as a stand-alone DVI-to-PDF converter. Incidentally, most of the `\special` features developed recently for viewing legacy DVI files (see below) have become permanent additions to the "black box" part of the DVIPDF program.

What shape will maturity bring to AcroDVI? Two options currently hold our attention.

In Lesenko (1997), it was proposed to build a DVIPDF plug-in for Acrobat Reader. With this approach, to view a given DVI file in Acrobat Reader, one would push its icon onto the Acrobat Reader icon rather than onto the DVIPDF icon. The plug-in architecture promises to promote portability of AcroDVI.

A second reasonable option would make AcroDVI an autonomous Windows application distinct from DVIPDF. This architecture promises to facilitate orchestration by AcroDVI of *multilateral* collaborations among DVIPDF, Netscape, Zip, Acrobat Reader, *Ghostscript*, and so on.

As soon as *Ghostscript/Ghostview* under Windows provide support for the key functions "Open Doc" and "Close Doc" of DDE, we will make available a new AcroDVI configuration that replaces Acrobat Reader by *Ghostscript/Ghostview*. It will probably be less luxurious than with Reader, but it will, in addition, accept EPS files.

Fonts

DVI files do not contain fonts — that is one basic reason why they are so compact. The question then arises: where are fonts for AcroDVI to come from? The best one can hope is that, in practice, enough Type 1 fonts will be in AcroDVI's expansible repertoire, which is based chiefly on B.K. Malyshev's BaKoMa Type 1 font collection, covering essentially all fonts commonly used in freely distributed electronic science publications.

Adobe's Type 1 is currently the only font format supported by DVIPDF; TrueType fonts are not accepted. Nor are Adobe Type 3 fonts allowed, bitmapped or not; Acrobat Reader would in any case handle them poorly.

The Adobe Type Manager, which first made screen viewing with scalable (vectorized) fonts a significant reality is *not* needed by AcroDVI since the relevant functions have been absorbed into Acrobat Reader.

On MathCD, there are just a few DVI files that call for commercial Adobe Type 1 fonts. DVIPDF will not currently handle these unless you have them installed in Type 1 format. Since many of these have acceptable TrueType versions preinstalled by Windows, more support for TrueType would be desirable.

Until then, we recommend Malyshev's own *DView* for such fonts. It offers essentially universal font support — although different graphics support.

Installing AcroDVI

AcroDVI (including DVIPDF) currently runs under all recent versions of Microsoft Windows (not under version 3.x). It is freely available on the Internet (see Resources).

Currently, both AcroDVI and DVIPDF are presented as a directory of approximately 1.5 megaoctets (Mo), not including the BaKoMa font collection, which is another few megaoctets. As for many Windows programs, an installer program is used.

The installed system is largely *autonomous* in that it requires only the prior presence of the Acrobat Reader (v. 4.0 or higher), and *non-invasive* in that it alters the behavior of nothing outside its own installed directory (currently called `dvipdf`). To deinstall it, one just deletes that directory.

Hopefully, this means that AcroDVI will be as simple to use as Acrobat Reader itself. For sophisticated users, there is an extensive configuration file to play with.

Performance testing

The following performance figures are for a 1997 PC with a Pentium I processor operating at clock speed 200 Mhz under Windows 95. For other Windows environments, a simple correction for clock speed should give a good first approximation to performance. The standard warning that "your mileage may vary" is appropriate. The programs, like vehicles, are extensively configurable, and the files, like terrain, are diverse.

For a typical mathematics article, the conversion to PDF format goes at about 15 pages per second, about 4 times greater than with Acrobat Distiller, or with *Ghostscript* in its PS2PDF mode. Comparison is relevant since it would be possible to publish compressed PostScript files without included fonts while giving Acrobat Distiller or *Ghostscript* access to the same BaKoMa font collection.

For its PDF output, DVIPDF does both font subsetting and stream compression. (The new compressed Type 1 font format has yet to be exploited by DVIPDF.) The efficiency of its default PDF output is thus respectable but not yet optimal. For example, it is comparable to that of the PDF files currently published by the American Mathematical Society, for the electronic research journal *ERA* (*Electronic Research Announcements*). However, by playing with the settings of Distiller, we were usually able to do better with Distiller, typically by a 3:2 ratio, particularly for small files. Do not rush to conclude that this ratio in favor of Distiller applies to all math journals. Indeed, the advantage swung in favor of DVIPDF for the next test by (not quite) a 2:3 ratio. It seems that both these PDF compilers could still reduce PDF bulk somewhat, in spite of many years of effort in this direction. From this point, however, we will focus on AcroDVI as a viewer of DVI, while ignoring its role as a compiler of PDF.

The DVI files used by AcroDVI are far less bulky than the PDF files used by Acrobat Reader. As evidence, here are a few examples from the first 1999 issue of the *Electronic Journal of Probability*, which added the PDF format compiled by Distiller to its web offerings in 1998:

Article	*Pages*	`.pdf`	`.pdf.gz`	`.dvi`	`.dvi.gz`	*Adv*
1	11	402	284	48	22	12.9
2	19	437	320	78	27	11.8
3	19	459	343	85	35	9.8
4	81	1162	960	412	154	6.2
5(?)	12	251	112	55	33	3.4

All file sizes are given in kilo-octets (Ko). Notice that DVI files regularly compress to about 40% of their original size while PDF files compress far less (as big internal chunks are precompressed). The last column of the table, the DVI advantage, gives the size ratio of compressed PDF files to compressed DVI files. This is an accurate measure of modem transfer speed ratios — whether the files are compressed or not — because during modem transfer, all material is compressed. The same ratio will be roughly the file size advantage of DVI files on a CD-ROM such as MathCD, which attempts to make the best use of available space. Indeed, a thoroughly precompressed form of PDF would be chosen for such a CD-ROM while the DVI files would probably be zip-compressed, along with any auxiliary graphics files.

The fifth article was anomalous in a number of respects. It had `\special` commands; there were two `.eps` figures, and these were complemented by their `.pdf` versions from Distiller, and the total of these graphics inclusions was less than 12 Ko of insertions (compressed). The explanation for PDF being only 3.4 times less efficient than DVI turned out, on investigation, to be mostly due to a common error in the production of PDF; namely, it was made with bitmapped TEX fonts, which perform disastrously in Acrobat Reader. When this is corrected, one can expect a PDF size similar to that of the first article.

Here are a couple of further examples, the shortest and longest available articles from a 1999 issue of *ERA*:

Article	*Pages*	`.pdf`	`.pdf.gz`	`.dvi`	`.dvi.gz`	*Adv*
1	3	105	84	12	5.5	15.3
2	12	248	215	66	27	8.0

These examples were reworked by us using well-tuned settings for Distiller (Windows version); the results (see below) are more flattering for the PDF format while leaving substantial advantage to DVI. Note that the *difference* between efficient and inefficient PDF is often many times greater than the total size of a DVI version.

Article	*Pages*	`.pdf`	`.pdf.gz`	`.dvi`	`.dvi.gz`	*Adv*
1	3	62	50	12	5.5	9.1
2	12	144	118	66	27	4.4

In the same vein, we note that the electronic journal *Geometry and Topology* (see `www.emis.de`) posts no TEX format whatsoever, just the Adobe formats PS and PDF. For their first 1000 pages the average PDF bulk per page is 10 Ko (fonts included) or about 8 Ko compressed. Thus, the DVI advantage would probably be somewhat less than 4. Expert use of PDF does make a big difference.

The modem bottleneck. Modem transfer speed is an important time factor. With a good telephone modem and a good line one can hope to get a transfer rate of about 5 Ko per second of compressed material. Now, a mathematics article in DVI format is about 2 Ko per compressed page, and thus the transfer rate is about 2.5 pages per second. This is about the speed at which one can scroll through the article with the 200 MHz PC used for these tests. Note that DVIPDF converts to PDF format at 6 times this speed. The time taken is perhaps time lost, but it is negligible.

With poor telephone lines or modems, or again congested web conditions, transfers that last more than a minute or two are likely to be broken; clearly the large PDF files are the ones at greatest risk, and with present web protocols, partial transfers are completely wasted.

Article transport costs. To get a very rough cost estimate, consider a mathematics article of 100 pages posted on the Internet and ultimately downloaded by a thousand readers (the typical number of subscriptions to a paper journal). Let is assume, to get an easily calculated figure, that everyone uses a contempory 56K baud modem with telephone charges of $2 per hour and in compensation let us neglect all other charges. With these figures, the telephone cost for delivering the article is about $22 for DVI format and between $70 and $250 for PDF format. Such figures suggest that use of DVI does reduce data transport costs significantly.

Improving the AcroDVI environment. First, the problem to be solved: in browsing the literature, it is not uncommon to look quickly at dozens of articles. This can quickly eat up many megaoctets of disk space if PDF format is involved. Now, one of Murphy's computing laws asserts that any hard disk that isn't new is surely nearly full, no matter what its capacity, since "data expands to fill any void". Thus, DVIPDF constantly risks running out of space.

To largely eliminate this overflow risk, there will be a setting for AcroDVI that makes the PDF file ephemeral and invisible. As soon as the next DVI file is processed, the previous invisible PDF will be erased. (That is no loss, since it can be regenerated quickly.) With this scheme, it suffices to verify at the beginning of a browsing session that your hard disk has enough space for the largest

single PDF file you expect to read, plus enough space for the relatively small DVI files.

Going one step further, the speed of AcroDVI can now be *doubled* by switching off compression of the the PDF output. At this point AcroDVI has been nicely optimized as a DVI viewer — at the cost of temporarily neglecting its role as a PDF compiler.

Comparing PDF and DVI formats

Adobe's Acrobat Reader has PDF as its native file format. This format is very autonomous:

- Graphics objects are always embedded within the PDF file.
- Fonts are usually embedded as well (the alternative, to use system fonts, has proved somewhat unreliable).

These positive features bring some disadvantages:

- Bitmapped graphics are unlikely to be displayed on-screen at optimum quality since that means no scaling. Vectorial graphics may not be seen in their full glory since that often requires the full screen.
- It is difficult to export graphics objects from the PDF file in the most useful formats.
- PDF files tend to be many times larger than DVI files. This is, of course, partly because of the font burden,[1] but the complexity of the PDF file structure brings substantial hidden costs.

Besides its space economy, the DVI format has other virtues worth mentioning. Like all of TEX, the DVI format is very stable, in spite of (and even because of) its `\special` appendages. This is important for archiving. Second, DVI is simple: only a few pages in Knuth's book on the TEX program (Knuth, 1986) are needed to define it adequately. Finally, one can derive from DVI all formats currently used for mathematics, excepting TEX source (i.e. the `.tex` file).

The strongest argument for PDF format has been the wide availability and high performance of the Acrobat Reader. Particularly outstanding are the user interface, the graphics quality, and the graphics speed. Adding to this: search, hypertext, copy-and-paste to text files, annotations, printing facilities, and PostScript (or EPS) export, it is clear that Acrobat Reader is a major contender for the affections of the reading public.

[1] The journal *Geometry and Topology* posts PDF format both with and without included fonts; omission of fonts economizes 25% over the first thousand pages of articles.

This does not prove that Acrobat Reader has no rivals among DVI readers. For example, xdvi (under unix) is by far the fastest viewer; the recent *BaKoMa DView* can do better in quality and scope of typography; emTEX provides better search and text export. Interesting new DVI viewers continue to appear: for example, *tkdvi* and *nDVI* (see Resources for details). It would be destructive not to serve such DVI readers. And ultimately destructive of TEX itself since TEX systems are typically built around them.

EPS, and now PDF, PNG and JPEG

The schemes to be described follow proposals in (Siebenmann, 1996); they are just some of many that have been elaborated for integration of graphics into the PDF output of DVIPDF (see Lesenko, 1997, 1998).

Let us begin by considering the graphics integration issue that arose for electronic journal articles to appear on MathCD. The DVI format is usually one of two or three presented, and it entails, for each article, one DVI file accompanied perhaps by some EPS graphics files. For MathCD it was important that the TEX version of the articles *not* be necessary for the integration of the reformatted graphics files.

Since DVIPDF cannot, on its own, convert EPS graphics to PDF, it was initially decided to provide PDF versions of the graphics objects via Distiller. One reason for this decision was that the PDF versions of vectorial graphics files are of optimal quality and often quite efficient, provided that font subsetting is used in creating them. Inasmuch as these PDF files can be immediately viewed by Acrobat Reader on most platforms, this conversion is of immediate benefit to almost all users.

Gradually, it became apparent that conversion to PNG and JPEG formats by various methods sometimes offers greater advantages. Fortunately, the solution to be described for PDF extends to PNG and JPEG graphics files.

The `\special` syntax in the DVI files used for EPS integration was most often the one used by Tomas Rokicki's `epsf.tex`. Aiming to exploit pre-existing DVI files using with Rokicki's *dvips*, we decided to have DVIPDF interpret the existing Rokicki syntax:

```
\special{Psfile=test.eps llx=11 lly=22
         urx=33 ury=44 rwi=550 rhi=660}
```

This is probably the world's most common `\special` syntax.[2]

The unit for the first four "bounding box" entries is 1 bp ("big point"). `llx` is the x-coordinate of the lower left corner of the bounding box, etc. Most often (but not always), this bounding box has simply been copied by TEX from the bounding box indicated in the EPS file header.

The last two entries, tagged by `rwi` (for real width) and by `rhi` (for real height), specify, in units of 0.1 bp, the width and height of the integrated bounding box on the output page. Either of these two entries, may be absent, in which case uniform scaling is used. By convention, the integrated box has its lower left corner placed at the DVI insertion point.

The (expanded) argument of this `\special` command is passed intact into the DVI file. Beware that it is normally generated inside of TEX, so that the author sees only some high-level commands, as those found in `epsf.tex`.

For both the `.eps` file and its derived `.pdf` file, the figure is located on a coordinate plane with unit of length = 1 bp; also, the scale and orientation are the same for both planes. In the event the `.pdf` file was created by *Ghostscript*, the two coordinate systems will be exactly the same. Then the *dvips* rules of integration from *dvips* are applied without modification and the results are identical.

If the `.pdf` file was created by Distiller, the two coordinate systems are related by a translation and some care is required required to make it predictable. We omit the details.

In fact, MathCD has used Distiller mainly, encountering only occasional problems. Fortunately, the reader of an article will be completely oblivious to such complications; only website editors, CD-ROM editors, and conscientious authors are concerned.

Generalizing to bitmapped graphics. There is an important variant of the above mechanism that is optimal for bitmaps. EPS and PDF are very general formats that can accommodate vectorial or bitmapped images; however, for bitmaps, EPS tends to be bulky and slow, and both seem to obstruct the recovery of embedded bitmaps. On the other hand, bitmap manipulation tools such as XNview under Windows and Linux/UNIX or *Graphics Converter* for Macintosh are easy to obtain and can generate an EPS format at any time. Thus, to the extent that you wish to grant full control of bitmapped graphics to the reader of your article, you may wish to use a native bitmapped norm.

The converse applies too: one can lock a PDF file or restrict its use in various ways. And it must be conceded that PDF manages to inherit the space efficiency of both leading public bitmapped formats: PNG and JPEG.

Recall that PNG (Portable Network Graphics) is the most efficient contemporary norm for faithful bitmap compression and is suitable for scientific figures and for most of the myriad uses which the commercial GIF format enjoys on the web. PNG is typically 15% more compact than GIF.[3] JPEG is the dominant "lossy" format for compression of low-contrast color images such as photos. JPEG (like GIF) is well supported by current versions of the Web browsers.

Hopefully, the above considerations will encourage more TEX users to exploit PNG and JPEG bitmaps. Those who are still restricted to vector graphics in TEX should be reminded, every time they see a web browser, that the full gamut of (still) bitmapped images, color included, as seen on the web, are begging to be used in TEX.

What we have said about PNG and JPEG being native or editable graphics formats is to some extent true for PDF. Indeed, the Windows graphics program Mayura Draw uses it as its storage format; however, it does not read arbitrary PDF files.

It is clear from the above conversion, that the preparation *ab initio* of a manuscript in DVI format with PNG bitmapped graphics inclusions can use the conventional EPS integration mechanism. We summarize since the process applies with little change also to JPEG and PDF:

- convert the PNG graphics to EPS, in XNview or similar program;
- integrate the `.eps` file using the consensual `special` command; and finally,
- replace the `.eps` file by the original `.png` file (the `.eps` file is usually bulky and is perhaps best discarded since it can be regenerated if the need arises).

The end user then just pushes the `.dvi` file onto the AcroDVI icon and viewing in Acrobat Reader will begin — using the original PNG graphics.

[2] It is not, however, the simplest for the job; indeed, one could get by without the "bounding box" entries (cf. Siebenmann, 1996).

[3] Unfortunately, the leading browsers, Netscape and Internet Explorer, have been tardy and half-hearted in their support for PNG. It may become necessary for AcroDVI to support GIF.

But there is a shortcut; it is unnecessary to generate an EPS file. Specifying

```
DoBBoxFile  =YES
```

in a configuration file, preview the PNG by pushing its icon onto that of AcroDVI. As a by-product, this previewing creates an auxiliary file (extension `.bb`), which contains the BoundingBox comment as in an EPS file header. With a suitable macro package such as `boxedeps.tex` (version for year 2000) or the LaTeX packages `graphics` or `graphicx`, the `.bb` file can be used *in lieu of* an EPS file.

Auxiliary roles for *Ghostscript*. The first is to allow on-the-fly integration by AcroDVI of EPS files into DVI format; optional settings of AcroDVI enable this when *Ghostscript* is present. This is very useful for viewing legacy DVI-plus-EPS postings prevalent on the Internet.

When an author or publisher is preparing an article for publication in DVI format with graphics inclusions, the strategy should be to vary the graphics format: maximize image quality while minimizing bulk. Effort spent on this often leads to surprising but useful results (see the 1999 documentation for `boxedeps`). Thus, it is advisable to urge authors to present originals of all graphics objects.[4]

Secondly, *Ghostscript* is a valuable converter to bitmap formats from PS, EPS, and even PDF; it has command-line options for parameters such as resolution. Unfortunately, *Ghostscript* has its quirks as a rasterizer. On the other hand, we have mentioned that the Adobe PS- and PDF-based systems seem loath to surrender internally stored bitmaps; they can be likened to a bank so eager for deposits that it has forgotten to provide for withdrawals. When need for withdrawals comes, *Ghostscript* may be your best friend.

The medium molds the message. One has to bear in mind that the various graphics formats and the various viewing mechanisms may influence what ultimately reaches the human eye. The pages of *TUGboat*, for example, are printed in black and white by photo-offset methods and will never faithfully render the colored iris that is the icon for AcroDVI.

For the reader's amusement, Fig. 1 shows in black-white several rather different renditions of the iris, all of which derive from one multicolored pastel original contributed by Tina and Keira Miyata. This

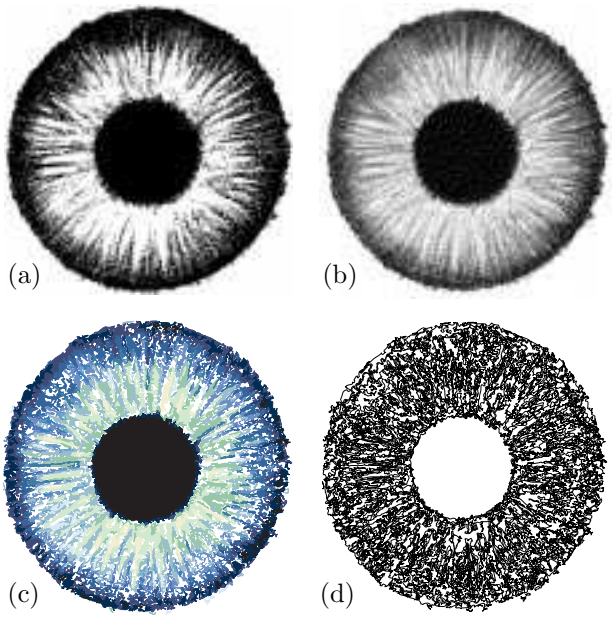


Figure 1

was scanned in 32-bit color to a 450×450-pixel bitmap, and stored as a 57 Ko file in JPEG format. This *TUGboat* article used the `.eps` versions.

(a) a suitable projection to black-white (= b/w) by *Graphics Converter*; size 30 Ko as `.eps.gz`, 20 Ko as `.pdf.gz`, 14 Ko as .png.

(b) is derived by Floyd-Steinberg filtering by *Graphics Converter*; size 36 Ko as `.eps.gz`, 26 Ko as `.pdf`, and 18 Ko as `.png`,

(c) (colored) arises from vectorization, by 16 regions of flat color, using Adobe's Streamline (v. 3); size 144 Ko as `.eps.gz`, 146 Ko as `.pdf.gz`, and 30 Ko as `.jpg`.

(d) (b/w) is the 1000 or so curves that are the boundaries of the 16 colors of (c); size 203 Ko as `.eps.gz`, 186 Ko as `.pdf.gz`, and 15 Ko as `.png`. This last bitmap blurred the curves; doubled resolution gave a 53 Ko `.png` file.

The quest for image clarity and beauty is an empirical art; with no testing, the results of photo-offset printing may be disappointing. We apologize in advance.

On the need for DVI efficiency

There is currently lukewarm support for the use of efficient methods. What can be done efficiently by astute programming is by preference done by the liberal expenditure of RAM or disk space or processor power. Thus, for AcroDVI to be taken seriously, cogent evidence is wanted that the resources economized through maintaining and

[4] For example, in preparing MathCD, the lack of such originals has been more of a vexation than the lack of TeX source files!

developing the efficient DVI format can be used decisively.

This is perhaps most evident with CD-ROMs. A CD-ROM contains about 650 Mo of data. If exploited to archive mathematics in compressed DVI form (and no other) a CD-ROM could contain about 300,000 pages of mathematics. That is enough space to record all the mathematics currently on the `xxx.lanl.gov` "e-print" archive (recently named "arXiv"), which is said to amount to about 200,000 pages. Alternatively, it is enough to distribute all the mathematics research articles published in one year (on paper *or* electronically). Again, it is enough space to reprint the whole of the *Annals of Math* (the most prestigeous math journal) plus the whole of *Crelle* (the oldest math journal).

Going beyond mathematics, it might be possible to present a complete encyclopedia on a single CD (or two), using compressed DVI (and graphics) files for storage and AcroDVI for viewing. Currently, the favored storage format for encyclopedias is RTF (Rich Text Format) and the usual viewer is MSWord. RTF enjoys efficiency comparable to that of HTML and DVI; it allows the same enviable flexibility of line length as HTML, and it is somewhat more expressive than HTML but less so than DVI. AcroDVI (allied with Acrobat Reader) offers the best typography and $peed.

Although such projects may not be realized in the immediate future, MathCD is intended to hint at them all.

There should also be evidence that CD-ROM capacity will not grow so fast that it outstrips the increasing demand for such permanent storage. If it does, then it is plausible that there is room for waste. The spectacular 1000-fold growth of the capacity of inexpensive hard disks in the last dozen years has fed wild expectations of storage technologies. But the reality for CD-ROMs is sobering. It is now known that the next (second) generation of CD-ROMs, called DVDs (Digital Versatile Disc) coming about 15 years after the first, will be based on a simple evolution of the current CD-ROMs: a rough doubling of density is involved, along with use of both sides of the disk. The capacity gain to 4.5 Gig[5] will be somewhat less than 10-fold (not 1000-fold). This is a factor frighteningly similar to the wastage factor that would be imposed by general adoption of the bulky PDF format. Furthermore, it could be a decade before the new CD-ROM format is sold at the affordable prices of today's CD-ROMs, since that is the time it took for today's CDs to reach mass consumer prices. This is one of the strongest arguments for retaining the efficient DVI norm. Fortunately, DVD readers will accept today's CD-ROMs.

The current pause in progress of telephone modem speeds gives additional arguments. The 56 kilobaud telephone modems of today are considered to be the last gasp of a tired technology up against what is called the "Shannon limit". In this case, a dramatic switch to ADSL (Asymetrical Digital Subscriber Lines) is being promoted with great speed gains: nominally 1.5 megabits/sec download and .5 megabits for upload (but, in practice, perhaps only one third or one quarter of that). There remains the question whether and when this technology will be as widely available and as affordable as the present modem technology.

Although hard disk capacity has been growing prodigiously, electronic libraries such as ELibMath EMS (`www.emis.de`) could come to need DVI's polyvalence and efficiency. Thus far, a hard disk of a few gigaoctets is sufficient to store the entire library of a few dozen journals. At current affordable prices for storage, dozens of mirror copies of the library have been established worldwide. As time passes, journals are not only multiplying and individually growing but are offering more and more formats for downloading, notably the bulky PDF. If and when this causes overflow of the current generation of the ELibMath hard disks, DVI format could offer an attractive remedy.

The `xxx.lanl.gov` e-print server has shown the way on economy by deriving essentially all formats from a `.tex` source on demand. This server successfully deploys immense expertise and resources under UNIX systems and manages to compile any document from `.tex` files to derive on-the-fly any other format the user requests. To do much the same on a CD-ROM, but using `.dvi` format, seems just within the realm of possibility — relying heavily on the greater simplicity and wide acceptance of DVI format. The first edition of MathCD will nevertheless be far more liberal (heteroclite) than the `xxx.lanl.gov` server.

We conclude that the economy and polyvalence of TEX's original DVI norm may indeed be the magical stuff from which dreams can be woven.

Is AcroDVI in the lead?

As a front end to Acrobat Reader for DVI viewing, how well does AcroDVI face competition?

[5] Double that for two-layer versions — whose durability is, unfortunately, in doubt.

There are several interesting *indirect* competitors that we merely mention in historical order: *Ghostscript/Ghostview*, then Distiller teamed with Acrobat Reader, and most recently pdfTEX (see Thành, 1998), also for use with the Reader.

Potentially, the strongest *indirect* competitor would be Acrobat Reader itself using a more compact and agile version of PDF format — but there is no sign of that.

One *direct* competitor of AcroDVI is Malyshev's *BaKoMa DView*, which not only has the broadest typographic capabilities in the TEX world of 1999, but also the ability to output PDF files. We leave the user to judge the relative virtues. Both will be provided on MathCD.

A second direct competitor is dvipdfm by Mark A. Wicks, which surfaced in 1998. It is an autonomous converter quite similar in concept to DVIPDF. Executable binaries are available on CTAN for 2 platforms, W9X/NT and i386 Linux. The reviewer of our paper informed us of many compiled dvipdfm binaries on the TEX-Live 4 CD-ROM. The platform/OS combinations served include: DEC alpha/OSF4, HP/HPUX10, i386/Linux, SGI/IRIX6.2, RS6000/AIX4.1.4, Sparc/Solaris 2.5–2.6, and Windows (32-bit).

Thus far, neither of these direct competitors has provided close integration with Acrobat Reader. It is probably fair to say that both are presently aiming at PDF publication, *not* DVI viewing. They are not yet competing frontally — but they soon could.

As for support of the most frequently used `\special` commands, *BaKoMa DView* is well advanced, thanks to adherence to *dvips* syntax. AcroDVI has some catching up to do here because it originally fashioned its own `\special` syntax; basic functionality for color and hyper-references are, however, present. Least adapted for viewing legacy DVI files is dvipdfm — because of its reliance on 'pdfmark' syntax; however, it has good basic `\special` functionality.

On the other hand, the recent wide porting of dvipdfm and distribution via the TEX-Live CD could well eclipse DVIPDF, and with it, AcroDVI. If that is our fate, we hope that both DVIPDF and AcroDVI will nevertheless be remembered as seminal proofs of feasibility.

Acknowledgements and History

The second author is grateful for an invitation from Stanislas Klimenko to visit IHEP in Protvino for several weeks in the autumn of 1997 to work with the first author, and also with Basil Malyshev. Basil has very kindly permitted us to distribute a version of his BaKoMa font collection with AcroDVI.

The idea of exploiting DVIPDF and Acrobat Reader together as a feature-rich DVI reader has been a subject of discussion between us (Lesenko and Siebenmann) since the 1996 TUG meeting in Dubna, Russia. For a long time, this project remained on a back burner while basic features of DVIPDF were perfected by the first author. As MathCD project took shape, it offered many stimulating design challenges, and the last year has brought substantial progress that seems to justify our early optimism.

Resources

Acrobat: a series of products by Adobe Inc., including Acrobat Reader, and Acrobat Distiller; the former is free while the latter is sold (but low prices for Distiller are available to academic users in many countries). Supported platforms include: Windows 3.x, Windows 9X, Windows NT, Macintosh, OS/2-Warp, Linux, IBM-AIX, SunOS, Solaris, SGI-IRIX, HP-UX, and Digital UNIX. Adobe's website address: `www.adobe.com`.
There is an active news list (`comp.text.pdf`) that can provide user support. See also EMJ, below, in particular Nelson Beebe's comments of 23 April 1999.

AcroDVI (with DVIPDF): by S. Lesenko and L. Siebenmann. Alpha versions are posted by anonymous `ftp` in Europe and N. America:
`topo.math.u-psud.fr/pub/tex/`
`cmstex.maths.umanitoba.ca/pub/acrodvi`
When AcroDVI is reasonably stable, it will be submitted to the CTAN archive.

BaKoMa TEX: by Basil K. Malyshev. A TEX implementation for the Microsoft Windows OS that appeared in 1998. Includes an advanced version of the BaKoMa font collection, the DVI viewer DView, and a DVI-to-PDF converter. Available from CTAN and from `ftp://ftp.mx.ihep.su`.

`boxedeps`: by Laurent Siebenmann. A macro package for EPS graphics integration that is valid for all PS printer drivers. Available from CTAN. The year 2000 version co-operates with some PDF compilers to integrate PDF, PNG, JPG graphics using `.bb` files.

dvips: by Tomas G. Rokicki; available from CTAN in the `dviware` directory.

dvipdfm: by Mark A. Wicks. A DVI-to-PDF converter that appeared in 1998:
`http://odo.kettering.edu/dvipdfm/`
Currently available on CTAN for i386 Linux, and for Windows 9X/NT as part of the MikTEX and fpTEX distributions.

EMJ: the Electronic Math Journals discussion list: `http://math.albany.edu:8800/hm/emj`.

`graphics`, `graphicx`: LaTeX2$_\varepsilon$ packages by David Carlisle and Sebastian Rahtz, on CTAN.

Graphics Converter: by Thorsden Lemke. bitmap editor and converter for Macintosh; shareware. `www.lemkesoft.de`.

Ghostscript: by Peter L. Deutsch.
A PostScript and PDF interpreter, that provides bitmapped or PDF output; the latter function is called PS2PDF.
`ftp.cs.wisc.edu/pub/ghost/aladdin`.

GSview: by Russell Lang. A viewer based on *Ghostscript*
`ftp.cs.wisc.edu/pub/ghost/rjl/`.

MathCD: CD-ROM (in prep.) devoted chiefly to journals and software for mathematics.
Go to `MathCD.html` at the editors' web sites: `www.math.washington.edu/~burdzy`, `topo.math.u-psud.fr/~lcs`, and `rsp.math.brandeis.edu`.

Mayura Draw: a graphics program by Karunakaran Rajeev; its native format is a dialect of PDF. `www.wix.com/mdraw210.zip`.

nDVI: a DVI viewer by K. Peeters. `norma.nikhef.nl/~t16/ndvi_doc.html`.

tkdvi: a DVI viewer by A. Lingnau. `www.tm.informatik.uni-frankfurt.de/~lingnau/tkdvi`.

XNview: by Pierre-E. Gougelet, bitmap editor and converter for Windows, Linux, etc.: `latour.univ-paris8.fr/~pierre`.

References

Bienz, Tim; Richard Cohn; and James Meehan. *Portable Document Format Reference Manual*. Addison-Wesley, Reading, Massachusetts, 1993.

Knuth, Donald. *TEX The Program*. Addison-Wesley, Reading, Mass., 1986.

Lesenko, Sergey. "The DVIPDF Program." *TUGboat* **17**(3), 252–254 (1996).

Lesenko, Sergey. "DVIPDF and Graphics." *TUGboat* **18**(3), 166–169 (1997).

Lesenko, Sergey. "DVIPDF and Embedded PDF." Proceedings of Euro-TEX Conference, St. Malo. *Cahiers GUTenberg* **28–29**, 231–241 (1998). `www.gutenberg.eu.org/pub/GUTenberg/`

Malyshev, Basil. "Problems of the conversion of METAFONT fonts to PostScript Type 1". *TUGboat* **16**(1), 60–68 (1995).

Siebenmann, Laurent. "DVI-based Electronic Publication." *TUGboat* **17**(2), 206–214 (1996).

Sojka, Petr, Hàn Thế Thành and Jiří Zlatuška. "The joy of TEX2PDF — Acrobatics with an Alternative to DVI Format." *TUGboat* **17**(3), 244–251 (1996).

Thành, Hàn Thế. "The pdfTEX Program." Proceedings of Euro-TEX Conference, St. Malo. *Cahiers GUTenberg* **28–29**, 197–210 (1998). `www.gutenberg.eu.org/pub/GUTenberg/`

Post-Conference Addendum

With reference to the discussion on modem downloading speeds (section "On the need for DVI efficiency"), Michael Doob reports top speeds near 500 Ko/sec on an optical cable network installed originally for cabled home television. This is stunning progress; even the authors' institutional ethernet LANs have never offered speeds quite so high. Curiously, the slower ASDL technology is attracting more investment. One should bear in mind that better Internet access may well increase 'peak time' Internet congestion, at which times effective throughput is often less than for a simple telephone modem.

We authors thank Michael Doob for hosting our alpha version, and also for making the oral presentation in Vancouver, when, at the last minute, the first author was unable to attend.

MathKit: Alternatives to Computer Modern Mathematics

Alan Hoenig
Department of Mathematics
John Jay College
445 West 59 St.
New York, NY 10019
(516) 385-0736 or (212) 237-8858
ajhjj@cunyvm.cuny.edu

Abstract

It is possible to generate hundreds of new math fonts using specially finagled math fonts produced by MetaFont to match Type1 PostScript fonts. This talk describes the *MathKit* project which enables authors ignorant of MetaFont, PostScript, and virtual fonts to create and use these fonts in a reasonably easy manner.

Introduction

I have long been impressed by the ingenuity and persistance of the TEX community as its members have gallantly shown how TEX can keep pace with all sorts of publishing needs and with all kinds of computer innovations, such as TEX and the World Wide Web. But I have long been struck by one apparent gap in this effort — there is no good way to typeset mathematics if you want to use any of the beautiful Type 1 PostScript fonts instead of Computer Modern. It is common to see authors embed Computer Modern math in Times Roman, say, but CM math is really too spindly for such typesetting to be as good as we know TEX is capable of. Several years ago, I wondered if there was a way to close this gap. One of the solutions I came upon is the subject of this talk. I'm particularly pleased by it because poky old METAFONT is an important component of this system. Perhaps *MathKit*, the name of my system, will help usher METAFONT into the next millenium.

MathKit is one attempt to deal with typesetting mathematics using fonts *other* than Computer Modern. Till now, authors have had few alternatives:

- They can use CM math together with a text font family such as Times Roman, but the result is not professional.
- They can use proprietary math fonts, such as MathTime or Lucida New Math, but that requires spending money.
- They can use the Euler math fonts, but these letterforms are a bit too idiosyncratic for some, and it is not well known how to properly implement them anyhow.

MathKit aids in the creation of math fonts which are compatible with a text font family — that is, it can help you typeset a Baskerville math document where the equations really look Baskerville-ish. Depending on your choice of parameters, you also get **bold** math fonts. *MathKit* consists of a perl script and some auxiliary files to help an author — even one ignorant of virtual fonts or of METAFONT — to perform these tasks.

What it does — a detailed look

MathKit takes METAFONT parameters that are appropriate to an outline font family and uses these to create new math fonts with METAFONT. The symbols and other special characters in these new fonts look pretty good — and are compatible with your outline fonts — but the italics and numerals look ghastly. Fortunately, that's not a problem. Using virtual fonts, we manufacture math fonts that combine the new special symbols (done by METAFONT that look pretty good) with letters and numerals from the outline fonts while we throw away all the ghastly stuff. *MathKit* does this work for you; it provides scripts for the remaining steps (all this is described below). It also provides style files for plain TEX and for the NFSS of LATEX for you to use these fonts in your documents. *You don't need to know anything about* METAFONT *or virtual fonts to use MathKit and the resulting fonts.*

This version of *MathKit* comes with three sets of font templates. Since Times Roman and Palatino are so common, I have prepared templates for these fonts. For fun, I have also prepared a template for Monotype Baskerville. Times comes in regular and bold series, Palatino is regular only and Baskerville in regular and semibold.

Unbound Orbits: Deflection of Light by the Sun

Consider a particle or photon approaching the sun from very great distances. At infinity the metric is Minkowskian, that is, $A(\infty) = B(\infty) = 1$, and we expect motion on a straight line at constant velocity V

$$\begin{aligned} b &\simeq r\sin(\phi - \phi_\infty) \simeq r(\phi - \phi_\infty) \\ -V &\simeq \tfrac{d}{dt}(r\cos(\phi - \phi_\infty)) \simeq \tfrac{dr}{dt} \end{aligned}$$

where b is the "impact parameter" and ϕ_∞ is the incident direction. We see that they do satisfy the equations of motion at infinity, where $A = B = 1$, and that the constants of motion are

$$\mathcal{J} = bV^2 \tag{1}$$

$$E = 1 - V^2 \tag{2}$$

(Of course a photon has $V = 1$, and as we have already seen, this gives $E = 0$.) It is often more convenient to express $\mathcal{J}$ in terms of the distance r_0 of closest approach to the sun, rather than the impact parameter b. At r_0, $dr/d\phi$ vanishes, so our earlier equations give

$$\mathcal{J} = r_0\left(\frac{1}{B(r_0)} - 1 + V^2\right)^{1/2}$$

The orbit is then described by

$$\phi(r) = \phi_\infty + \int_r^\infty \left\{ \frac{A^{\frac{1}{2}}(r)\,dr}{r^2\left(\frac{1}{r_0^2}\left[\frac{1}{B(r)-1+V^2}\right]\left[\frac{1}{B(r)-1+V^2}\right]^{-1} - \frac{1}{r^2}\right)^{\frac{1}{2}}} \right\}.$$

The total change in ϕ as r decreases from infinity to its minimum value r_0 and then increases again to infinity is just twice its change from ∞ to r_0, that is, $2|\phi(r_0) - \phi'_\infty|$. If the trajectory were a straight line, this would equal just π;

$$\Delta\phi = 2|\phi(r_0) - \phi_\infty| - \pi.$$

If this is positive, then the angle ϕ changes by more than $180°$, that is, the trajectory is bent *toward* the sun; if $\Delta\phi$ is negative then the trajectory is bent away from the sun.

Figure 1: Here are Baskerville-like math fonts, produced by *MathKit*, together with Baskerville text fonts.

However, I have had excellent luck matching one of the templates with a non-related text family. The Baskerville-like template works very nicely with Monotype Janson and Adobe Caslon, for example. Consequently, it is possible to generate not three new math font families, but *hundreds* of them, as the title to this document proclaims.

What you get as final output

MathKit itself produces lots and lots of scripts and batch files. Once these are properly executed you get the following:

1. Detailed instructions, both onscreen and in a small ASCII file, telling you how to proceed.
2. Virtual fonts for math and text typesetting. You will also get fonts for bold math if a template containing bold parameters is supplied.
3. Style files for plain TEX and LATEX (NFSS). These files support bold math if bold parameter templates were present.

What you will need

All files can be found on any CTAN or mirror site, unless otherwise noted.

1. First off, you will need current versions of TEX and METAFONT. They must be sufficiently recent to support virtual fonts.
2. `fontinst`, version 1.5 or better. To install this software, retrieve *all* files from the

   `fonts/utilities/fontinst/inputs`

   area.
3. For plain TEX, Damian Cudgley's `pdcfsel` font selection macros are required. These can be found in `macros/plain/contrib/pdcmac`.
4. Perl needs installation as well: version 5 of Perl, a freely-available utility for all computer platforms and easily obtained from many computer archives and CD-ROM software collections. This is simply a matter of placing the `perl` executable somewhere on your computer's search path.
5. Your text fonts need to have been installed using Karl Berry's fontnaming conventions. Furthermore, these fonts must have been installed following the original TEX encoding, often denoted as `OT1` or `ot1`.
6. Working copies of the TEXware utilities `tftopl` and `vptovf`, which should already be part of your TEX installation. Make sure both these executables are in some part of your computer's search path.

Installation

Installation of *MathKit* consists of three steps:

1. Create a directory called `mathkit`, and install all the *MathKit* files in it.
2. Create a work directory below `mathkit`; switch to this directory to do all your work.
3. Finally, there are a few parameters that need *careful* adjustment at the beginning of the file `mathkit.par`. Check the documentation for more details.

MathKit also makes it possible to typeset with some special font types, including blackboard bold, calligraphic, Fraktur, typewriter monospaced, and sans serif, and will provide typesetting commands for these fonts, provided the latter exist. Except for sans serif, though, you have relatively little choice in which kinds of fonts to install. These fonts and font sources are all available on CTAN. Here's what *MathKit* expects:

- *MathKit* uses the calligraphic alphabet in the Computer Modern symbol fonts.
- The typewriter font must be installed under the name `cmtt10` and you will need an outline form of this font.
- You will need the `eufm10` Euler Fraktur font in outline form for Fraktur typesetting.
- You will need the METAFONT source for Alan Jeffrey's blackboard bold fonts for blackboard bold typesetting. On CTAN, these can be found in the `fonts` area, or perhaps `fonts/bbold`.
- You have much greater freedom for sans serif fonts, as discussed above.

Executing the software

The main *MathKit* script requires three parameters at the command line:

1. The name of the parameter template: `tm` refers to Times-like parameters, `pl` to Palatino-like, and `bv` to Baskerville-like.
2. The name under which text fonts are installed. This is apt to be something like `ptm` or `mnt` for Adobe Times or Monotype Times New Roman, `ppl` for Palatino, and `mbv` for Monotype Baskerville (which is *quite* different from ITC New Baskerville). As mentioned above, though, you are welcome to any properly installed text font family as well. Simply specify its `fontname` abbreviation at the command line.
3. The encoding your fonts follow. Only `OT1` (original TEX encoding) or maybe `ot1` are currently allowed. Use `ot1` if your system doesn't allow uppercase file names.

Unbound Orbits: Deflection of Light by the Sun

Consider a particle or photon approaching the sun from very great distances. At infinity the metric is Minkowskian, that is, $A(\infty) = B(\infty) = 1$, and we expect motion on a straight line at constant velocity V

$$\begin{aligned} b &\simeq r\sin(\phi - \phi_\infty) \simeq r(\phi - \phi_\infty) \\ -V &\simeq \tfrac{d}{dt}(r\cos(\phi - \phi_\infty)) \simeq \tfrac{dr}{dt} \end{aligned}$$

where b is the "impact parameter" and ϕ_∞ is the incident direction. We see that they do satisfy the equations of motion at infinity, where $A = B = 1$, and that the constants of motion are

$$J = bV^2 \tag{1}$$

$$E = 1 - V^2 \tag{2}$$

(Of course a photon has $V = 1$, and as we have already seen, this gives $E = 0$.) It is often more convenient to express J in terms of the distance r_0 of closest approach to the sun, rather than the impact parameter b. At r_0, $dr/d\phi$ vanishes, so our earlier equations give

$$J = r_0\left(\frac{1}{B(r_0)} - 1 + V^2\right)^{1/2}$$

The orbit is then described by

$$\phi(r) = \phi_\infty + \int_r^\infty \left\{ \frac{A^{\frac{1}{2}}(r)\,dr}{r^2\left(\frac{1}{r_0^2}\left[\frac{1}{B(r)-1+V^2}\right]\left[\frac{1}{B(r)-1+V^2}\right]^{-1} - \frac{1}{r^2}\right)^{\frac{1}{2}}} \right\}.$$

The total change in ϕ as r decreases from infinity to its minimum value r_0 and then increases again to infinity is just twice its change from ∞ to r_0, that is, $2|\phi(r_0) - \phi'_\infty|$. If the trajectory were a straight line, this would equal just π;

$$\Delta\phi = 2|\phi(r_0) - \phi_\infty| - \pi.$$

If this is positive, then the angle ϕ changes by more than 180°, that is, the trajectory is bent toward the sun; if $\Delta\phi$ is negative then the trajectory is bent away from the sun.

Figure 2: Palatino-like fonts with Palatino text.

For example, I type

```
perl ../mathkit tm ptm OT1
```

in my work directory to create Times-like fonts following the original TEX encoding. To create my Janson/Baskerville fonts, I type

```
../mathkit bv mjn OT1
```

at the command line.

Currently, you get bold math fonts *unless* you choose the Palatino-like `pl` template.

Making the fonts

The following steps complete the font creation. Perform them all within the *MathKit* work directory (Step 2 in the "Installation" section).

1. Execute the file `makegf.bat` to have METAFONT create your pixel fonts. This step will take some time.
2. You will need to compress all the pixel files. Inside UNIX, you can do this via a series of commands such as

   ```
   foreach X (*.600gf)
   foreach? gftopk $X $X:r.600pk
   foreach? end
   ```

   Not all operating systems are so accommodating, so there is a file called `makepk.bat` which may be helpful in this regard. **Caution:** before executing this script, it will almost surely be necessary to edit it.
3. Execute the script `makepl.bat` to create some property list files needed by TEX.
4. Run the file `makevp.tex` through TEX. That is, execute the command `tex makevp` or something appropriate for your system. This step will take lots of time. Along with lots of superfluous files, this creates many virtual property list files with extension `.vpl`.
5. Create the actual virtual files by running every `.vpl` file through the program `vptovf`. You can do this easily in UNIX:

   ```
   foreach X (*.vpl)
   foreach? vptovf $X $X:r.vf $X:r.tfm
   foreach? end
   ```

   Even easier — execute the file `makevf.bat` that *MathKit* creates for you.
6. Test your fonts by processing `testmath.tex` (for LATEX users) or `testmatp.tex` (plain TEX) and then printing it. If adjustments are necessary, return to step 4 (run `tex makevp`) and proceed from that point onward. Adjustments to your fonts will be discussed below.
7. Only when you are completely satisfied with your new fonts should you execute the script `putfonts.bat`, which moves font files and style files to their proper places.

That still leaves behind files with extensions `.log`, `.mtx`, `.pl`, `.vpl`, `.bat`, `.600gf` (or something similar), and several other miscellaneous other files. You may safely delete all these.

Fine tuning and character adjustment The only adjustment that should be necessary are spacing adjustments to improve the appearance of over-the-character accents, subscripts, and character placement. The two test files that enable you test this are `testmath.tex` (for LATEX) and `testmatp.tex` (plain). Run one of these files through TEX and examine the printed output carefully. *MathKit* will have made two or more adjustment files for you that facilitate making changes to character spacing.

Font mongers note: you may be able to fine-tune the characters themselves by adjusting the parameter values in the *template* files to other than those provided. Feel free! If you find a particularly fine set of values different from what I have provided, I would be grateful if you passed them along to me.

Using your new fonts

MathKit produces two style files, one for LATEX and one for plain. Their file names are formed according to the naming scheme

$$\texttt{z}\langle\textit{mock-family}\rangle\langle\textit{font-family}\rangle$$

Here, ⟨*mock-family*⟩ is the two-character designation for one of the font parameter templates (such as `tm`, `pl`, or `bv`); the word 'mock' refers to the fact that these fonts imitate but don't equal the actual fonts in this family. ⟨*font-family*⟩ is the Berry family designation. Thus, if I create a Times-like set of fonts for use with font family `ptm`, I would find files `ztmptm.sty` (LATEX) and `ztmptm.tex` (plain). In the same way, the style files for mock-Palatino and mock-Baskerville fonts are named `zplppl` and `zbvmbv` (with the appropriate extensions). Style files for my Baskerville/Janson math fonts have names beginning with `zbvmjn`.

Plain TEX At the top of your file, include the statement

```
\input ztmmnt
```

(or whatever the style file name is). Then, standard font nicknames such as `\it` and `\bf` and the math toggles `$` and `$$` will refer to these new fonts.

If bold fonts have been generated, a command `\boldface` typesets everything in boldface — prose,

Unbound Orbits: Deflection of Light by the Sun

Consider a particle or photon approaching the sun from very great distances. At infinity the metric is Minkowskian, that is, $A(\infty) = B(\infty) = 1$, and we expect motion on a straight line at constant velocity V

$$\begin{aligned} b &\simeq r\sin(\phi - \phi_\infty) \simeq r(\phi - \phi_\infty) \\ -V &\simeq \tfrac{d}{dt}(r\cos(\phi - \phi_\infty)) \simeq \tfrac{dr}{dt} \end{aligned}$$

where b is the "impact parameter" and ϕ_∞ is the incident direction. We see that they do satisfy the equations of motion at infinity, where $A = B = 1$, and that the constants of motion are

$$J = bV^2 \tag{1}$$

$$E = 1 - V^2 \tag{2}$$

(Of course a photon has $V = 1$, and as we have already seen, this gives $E = 0$.) It is often more convenient to express J in terms of the distance r_0 of closest approach to the sun, rather than the impact parameter b. At r_0, $dr/d\phi$ vanishes, so our earlier equations give

$$J = r_0\left(\frac{1}{B(r_0)} - 1 + V^2\right)^{1/2}$$

The orbit is then described by

$$\phi(r) = \phi_\infty + \int_r^\infty \left\{ \frac{A^{\frac{1}{2}}(r)\,dr}{r^2\left(\frac{1}{r_0^2}\left[\frac{1}{B(r)-1+V^2}\right]\left[\frac{1}{B(r)-1+V^2}\right]^{-1} - \frac{1}{r^2}\right)^{\frac{1}{2}}} \right\}.$$

The total change in ϕ as r decreases from infinity to its minimum value r_0 and then increases again to infinity is just twice its change from ∞ to r_0, that is, $2|\,\phi(r_0) - \phi'_\infty\,|$. If the trajectory were a straight line, this would equal just π;

$$\Delta\phi = 2|\,\phi(r_0) - \phi_\infty\,| - \pi.$$

If this is positive, then the angle ϕ changes by more than 180°, that is, the trajectory is bent *toward* the sun; if $\Delta\phi$ is negative then the trajectory is bent away from the sun.

Figure 3: Times Roman math + Times Roman text.

mathematics, whatever. Bold math may be appropriate for bold captions, sections heads, and the like. Like any other font-changing command, this command should be placed within grouping symbols.

LaTeX and NFSS You simply need to include the style name as part of the list of packages that you use in the document. Thus, a typical document would have a statement like

```
\usepackage{ztmptm,epsf,pstricks,...}
```

at the outset.

If *MathKit* has created bold math fonts for you, a `boldface` environment will typeset everything in that environment as bold, including all mathematics.

Math support for other font families

The parameters for the font families are contained in files with names like `tm.mkr`, `tm.mks`, or `tm.mkb`. The extensions refer to "*MathKit* regular", "*MathKit* semibold", or "*MathKit* bold" sets of parameters. The current *MathKit* assumes that you will be creating at most one of the set of bold or semibold fonts but not both.

It was surprisingly easy to prepare these parameter files. I prepared a test document in which individual characters were printed on a baseline at a size of 750 pt. It's (relatively) easy to measure the dimensions of such large characters and METAFONT can be asked to divide by 75 to compute the proper dimension for ten-point fonts. It was particularly easy for me to make these measurements as I used Tom Rokicki's superior implementation of TeX for NeXTStep. This package contains on-screen calipers, which take all the work out of this chore.

If you plan to create your own parameter files for other font families, please use the supplied files as models. Make sure all measurements are given in terms of sharped points `pt#`;[1] *MathKit* looks for this string. And please consider placing this information on CTAN.

Other details; in conclusion ...

For additional information, please see my book, *TeX Unbound*. Sample output using *MathKit*-tweaked fonts appears throughout this article. The current version of *MathKit* is in the `fonts/utilities/mathkit` area of CTAN. Additional details concerning *MathKit* can be found in the documentation file `mathkit.tex`, part of this package.

Interested authors may care to investigate the author's companion package, MathInst, the current version of which appears in the `fonts/utilities/mathinst` area of CTAN. In case you have MathTime, Lucida New Math, Euler, or Mathematica math fonts, MathInst provides scripts for installing these fonts with text fonts of your choice.

This software is issued as is, subject to the usual GNU copyleft agreement.

If you have any questions, comments, or bug reports, please send them along to me.

References

[1] Bouche, Thierry. "Diversity in Math Fonts." *TUGboat* **19**,2 (1998), pp.121–135.

[2] Hoenig, Alan. "Alternatives to Computer Modern Mathematics." *TUGboat* **19**,2 (1998), pp. 176–187.

[3] Hoenig, Alan. *TeX Unbound: LaTeX and TeX Strategies for Fonts, Graphics, and More*. New York: Oxford University Press, 1999.

[4] Horn, Berthold. "Where Are the Math Fonts?" *TUGboat* **14**,3 (1993), pp. 282–284.

[5] Knuth, Donald E. *The METAFONTbook*. Reading, Mass.: Addison-Wesley, 1986.

[1] 'Sharped points' are "'true' units of measure, which remain the same whether we are making a font at high or low resolution" (*The METAFONTbook*, p. 33). See also pp. 32–35, 91–99, 102–103, 268, and 315 — all in *The METAFONTbook*.

Unbound Orbits: Deflection of Light by the Sun

Consider a particle or photon approaching the sun from very great distances. At infinity the metric is Minkowskian, that is, $A(\infty) = B(\infty) = 1$, and we expect motion on a straight line at constant velocity V

$$b \simeq r\sin(\phi - \phi_\infty) \simeq r(\phi - \phi_\infty)$$
$$-V \simeq \tfrac{d}{dt}(r\cos(\phi - \phi_\infty)) \simeq \tfrac{dr}{dt}$$

where b is the "impact parameter" and ϕ_∞ is the incident direction. We see that they do satisfy the equations of motion at infinity, where $A = B = 1$, and that the constants of motion are

$$J = bV^2 \quad (1)$$
$$E = 1 - V^2 \quad (2)$$

(Of course a photon has $V = 1$, and as we have already seen, this gives $E = 0$.) It is often more convenient to express J in terms of the distance r_0 of closest approach to the sun, rather than the impact parameter b. At r_0, $dr/d\phi$ vanishes, so our earlier equations give

$$J = r_0\left(\frac{1}{B(r_0)} - 1 + V^2\right)^{1/2}$$

The orbit is then described by

$$\phi(r) = \phi_\infty + \int_r^\infty \left\{ \frac{A^{\frac{1}{2}}(r)\,dr}{r^2\left(\frac{1}{r_0^2}\left[\frac{1}{B(r)-1+V^2}\right]\left[\frac{1}{B(r)-1+V^2}\right]^{-1} - \frac{1}{r^2}\right)^{\frac{1}{2}}} \right\}.$$

The total change in ϕ as r decreases from infinity to its minimum value r_0 and then increases again to infinity is just twice its change from ∞ to r_0, that is, $2|\phi(r_0) - \phi'_\infty|$. If the trajectory were a straight line, this would equal just π;

$$\Delta\phi = 2|\phi(r_0) - \phi_\infty| - \pi.$$

If this is positive, then the angle ϕ changes by more than 180°, that is, the trajectory is bent *toward* the sun; if $\Delta\phi$ is negative then the trajectory is bent away from the sun.

Figure 4: MathKit makes possible math bold typesetting. Here is is Times Roman bold math + Times Roman bold text.

fpTEX: A teTEX-based Distribution for Windows

Fabrice Popineau
Supélec
2 rue E. Belin
F-57070 Metz
France
fabrice.popineau@supelec.fr
http://www.ese-metz.fr/~popineau

Abstract

This paper deals with the ins and outs of porting the widely used teTEX distribution to the Windows environment. The choices made and difficulties experienced are related, a brief description of this huge distribution is given and the future work is sketched out.

Motivation

Context. More and more people need to use some sort of Microsoft environment, perhaps because of office suite or the like, or because of management staff decisions. Some of the greatest pieces of software have been developed on UNIX (or other operating system) well before the general availability of Windows. Thus, software such as TEX should be available on Microsoft operating systems, natively ported, and compatible with implementations on other operating systems. TEX by itself is largely platform independent but achieving complete compatibility for an entire distribution is better.

The Web2C TEX distribution is one of the greatest TEX implementations and has support for portability. Moreover, Web2C is the base of the widely used teTEX distribution for UNIX. Now that UNIX and Windows can easily share files across the network, thanks to tools such as Samba, it is most desirable to have not a teTEX-*like* TEX distribution for Windows, but the *actual* teTEX distribution for Windows.

Free TEX for Windows. In the fall of 1997, when I began to port Web2C to Win32, one main TEX distribution was available to Windows users: emTEX. This is still a great TEX environment but it was designed for MS-DOS first and then for OS/2. So, when Windows became 32-bit-aware,[1] those MS-DOS applications could not benefit from the new 32-bit flat mode, or at least not optimally. The so-called *DOS-extenders* were not as smart[2] as they are today. Some of the nicest features of emTEX, such as its `dvipm` previewer, were not available to Windows users. Moreover, emTEX's author, Eberhard Mattes, never released his sources.

At the same time, MiKTEX began to mature. Christian Schenk, author of MiKTEX, has followed a different way. He has designed a completely new Win32-oriented TEX distribution. Looking at his work, I questioned the usefulness of porting Web2C to Win32, but there were some reasons to do so:

Compatibility. Having a Windows TEX distribution based on exactly the same files as the UNIX one means you can *share* resources. For example, you can not only share `texmf` trees across the network but also configuration and format files.

Portability. Most of the ongoing developments around TEX are done with UNIX Web2C (see pdfTEX or ε-TEX). Being able to share source files means less efforts to compile a new release. There is another consequence: on several occasions, it has proven to be useful to compile the source code on something really different from UNIX. Errors that do not show up on one platform may do so on another one.

Usability. Lots of people are familiar with teTEX under UNIX. Having the very same distribution under Windows is a plus.

The plan. The porting tasks can be divided as follows. Note, however, that the job was not formally planned at all since it had to be done using mostly spare time. So the project has followed a circular technique, with some issues only being resolved quite recently. Below is a very short description of the next sections.

[1] In fact, even if Windows 9x can run 32-bit mode applications, only the Windows NT incarnation of Windows is a true 32-bit environment; see section about the previewer

[2] For example, support for *long filenames* was not available at first.

Command-line programs. The first goal was to have a `tex.exe` running under the Win32 API (Application Programming Interface), the set of functions that implement the `kpathsea` library.

Compilation environment. teTEX uses a powerful tool called `autoconf`. This tool relies heavily on having a UNIX shell and lots of UNIX utilities such as `sed`, `grep`, `awk`, and so on. Clearly, this is not something easy to find and run under Windows.

Shell issues. Moreover, the source distribution uses some shell scripts at *run-time*. It is not wise to suppose that the end-user will have a UNIX shell on a Windows machine.

Operating system-specific. Some issues like finding a replacement for file links or naming files on the network have been solved recently. The question of using the registry is also mentioned.

Previewer. No TEX distribution would be complete without a DVI previewer. So the port of xdvi was contemplated.

Installation. This is the trickiest part. Binary distributions were not common under UNIX but they are under Windows, and the installation process is very different.

Configuration. The process of configuring Web2C is very simple because it consists mainly of editing text files or setting up environment variables. The teTEX distribution introduces a smart tool to administer the system, and this task can be rendered in a Windows-oriented way as well.

Future work. There are many points that can be enhanced and some will be done in the very near future.

However, if tasks such as editing a text file, setting environment variables or unpacking an archive are usual in the UNIX world, they are not usual anymore in the Windows, world where end-users expect automatic or point-and-click things to happen. So the installation and configuration parts are very specific to Windows.

The contents of the distribution

Before discussing the porting issues, here is a brief outline of what is in the distribution:

- Web2C base distribution: TEX, METAFONT, METAPOST, DVIware and fontware tools
- each TEX extension or package that is found in teTEX:
  - ε-TEX, pdfTEX, Ω (Omega)
  - `dvipsk` and `dviljk` to print DVI files
  - `gsftopk` and `ps2pk` to rasterize Type 1 fonts to PK files
  - `mktex*` support programs for generating missing font files and `fmtutil` for building formats
- a DVI file viewer based on xdvi, but adapted to Windows
- packages found on the TEX-Live CD such as:
  - `dvipdfm`, to convert DVI files to PDF
  - `tex4ht` and `tth`, to convert TEX files to HTML
  - extra tools to deal with either DVI files, PostScript files or fonts.
- extra packages found only on the Win32 section of the TEX-Live CD:
  - `ttf2pk` and `ttfdump` will handle TTF fonts
  - `hbf2gf` will handle East-Asian fonts
  - `gzip` and `jpeg2ps`, which can be handy
- the teTEX `texmf` tree, which is not the least important part!

Command-line programs

The process. The Web2C distribution integrates all TEX-related tools around one main library called `kpathsea`. It was devised by Karl Berry to face the growing number of environment variables needed to set up a complete TEX distribution. Instead of setting environment variables, the path values and many other constants are looked for in a configuration file. This guarantees extensibility and is far easier to maintain.

The second point is the process of compiling TEX itself. The Web2C distribution owes its name from the Web → C translator that converts the original Web code to C programs. Basically, the following tools were needed:

- a C compiler targetting the Win32 API and, if possible, supporting the standard C library functions;
- some UNIX tools such as `sed`, `grep` and `awk`;
- the Perl language, which has proven to be useful to put glue between many parts of the building process, due to the lack of a shell with real programming capabilities under Windows.

Choosing a compiler. The availability of GCC — or rather of a native Win32 GCC — under Windows is quite recent. Moreover, GCC in its Cygwin[3] incarnation has some drawbacks under Windows:

[3] Most of the GNU tools have been ported to Win32 by *Cygnus Software* and their port is under the GNU Public Licence. See `http://sourceware.cygnus.com/cygwin/`.

- Every program is linked to a DLL (Dynamically Linked Library) that emulates UNIX calls; this slows down somewhat programs doing intensive file system calls,[4] for example.
- At the time I began the Web2C port, this DLL was not stable at all.
- Benchmarks on the same computer using GCC under Linux and the Microsoft compiler under NT have shown up to 20% less time on the same runs in favor of the Microsoft compiler.[5]

Thus, the Microsoft compiler was chosen. The general philosophy was to stick to the Win32 API as much as possible and avoid any layer to handle the translation, which might alter performance.

Many of the auxiliary tools needed for the build process were available either through the GNU-Win32 Cygnus project or from previous ports to MS-DOS; however, almost none of them were available natively ported to Win32. While I was at porting `kpathsea` and Web2C to Win32, I also adapted the tools I needed to Win32. This resulted in an archive of UNIX tools, many compiled by myself and the others gathered from the net. This archive is available in the same directory as fpTEX, in the CTAN archives.

Compiling `tex.exe`. The `kpathsea` library already had support at the source level for other platforms than UNIX, namely Amiga and VMS. So the path was already laid out. Fortunately, `kpathsea` already encapsulated almost all system calls needed for TEX. This was a great feature of the TEX source code, to precisely identify system dependencies.

Disks. The Windows environment knows about *device names* attached to disks whereas the directory tree structure under UNIX hides them.

Paths. The path separator is not the same but, fortunately, the Win32 API support '\' and '/' path separators.

Links. There are no hard or symbolic links under Windows.

Permissions. The permissions on files for UNIX and Windows are handled in a completely different way.

Some of the problems met were specific to Windows 9x, where standard C library calls are available but buggy. For example:

- `system()` is meant to run external commands but fails to return their exit code — it always returns true.
- `popen()` is available only for command-line programs but fails for graphical programs (the previewer uses this call!).
- `stat()` fails to recognize directories if their name has a trailing `/`.

All these problems have workarounds using the Win32 calls instead of the standard C calls.

Compilation environment

In order to make the build process safer and closer to what happens under UNIX, a number of decisions had to be made.

Makefiles. Every UNIX `Makefile` comes in a generic shape, `Makefile.in`, that needs to be instantiated by the complex process of `autoconf`. The UNIX `Makefile.in` is assembled and processed by the `m4` macro processor to generate the actual `Makefile` that will fit your own configuration. Moreover, those `Makefile` use many UNIX constructs (shell or other tools). So they are not usable as-is under Windows, where the process is somewhat different.

The Windows `Makefile` is built by hand from the UNIX `Makefile.in`. All common parts are stored in a special place. An initial `configure.pl` Perl script allows one to:

- configure the common parts with options like root of the destination directory, root of the source tree, and so on;
- ensure through the configuration that only the files generated by the current build will be referred to; that is, no external `kpathsea.dll` will interfere, no external `pdftex.exe` or `texmf` will be referred to when generating documentation or file formats;
- save and restore each of the `Makefile` files in a safe place.

Source code configuration. The same Perl script also undertakes the translation process of every `config.in` or `c-auto.in` configuration file into their definitive form. Since there is only one target operating system, there is no need to guess if the features are supported — just consult a table of features. Thus, doing this ensures better compatibility with the original source code.

Overall build process. The overall build process is done by another `build.pl` Perl script. This script delegates to the different `Makefile` and can be asked to:

- clean up the source tree at different levels
- rebuild dependencies
- build and/or install the whole release

[4] The `kpathsea` library can do lots of `stat` calls on a huge `texmf` tree.

[5] This was GCC 2.7.1 versus VC++ 5.0; the situation may have changed today.

- use different compilation modes, such as *debug* or *release*, *statically* or *dynamically* linked executables
- prepare for specific tasks, such as *profiling* or using advanced debugging and checking tools such as BoundsChecker
- install everything *from scratch*, including installing the latest teTEX `texmf` tree

Up to now, the build process is not clean enough, but nonetheless the source tree has been used successfully by people with no previous knowledge. Cleaning up the Win32 part of the source tree would allow more people to access it and contributions from the net could be expected.

Shell issues

The Web2C distribution may ask for font generation at run-time. This is done through the `kpathsea` library calling an external command when it fails to find some needed font.

Because of the complex and evolving nature of this process — how many versions of those `mktexpk` scripts have been devised? — the generation of fonts has traditionally been handled by shell scripts.

The shell requirement needed to be removed under Win32 and the Perl alternative, despite some drawbacks, was considered:

- Perl provides greater portability across such different operating systems as UNIX and Windows;
- Perl is not widespread enough under Windows and under UNIX, which means you can easily find it but not every single user will have it or want it;
- Perl has quite a large disk space footprint.

Had UNIX users been ready to switch from shell scripts to Perl scripts for this task, things might have been different but that was not the case. So the only risk-free and simple solution from the user point of view was to code the shell scripts in C. Given the complexity, the C version of the scripts required several rewritings before becoming reliable enough. But now, they do behave like their original shell counterparts. And the C version of these scripts can even be used under UNIX. The several `mktex*` shell scripts are provided as one DLL, with several stubs,[6] following the same philosophy as for the TEX engines — see next section. This means that `kpathsea` could be linked with this `mktex.dll` and could avoid creating a new process to generate a new font file.

[6] A *stub* is a small executable program linked to some DLL and whose only function is to set up some parameters before calling the DLL. This way, the same large DLL can be called in different ways by using only small executable programs.

Operating system-specific

Two main features have been added and one has been avoided.

No file links under Windows. The Web2C distribution uses file links under UNIX for linking programs under different names. The problem is that you can have several format files generated by one engine. For example, `latex.fmt` and `plain.fmt` are both run with the `tex.exe` engine. Under UNIX, the `tex` engine maybe linked under the names `latex` and `plain`, and the name under which the engine is linked determines which file format is loaded.

There are no file links under Windows [7] and all you can do is simply copying the `tex.exe` engine to `latex.exe` and so on — at the expense of disk space. Given the number of engines, some of them being quite large, it is important to overcome the problem of file links.

Fortunately, for executable programs, there is a natural way of doing something similar to file links using the Win32. The trick is to build a DLL with all the engine code and to have a small stub linked to the DLL. This way, the DLL is shared and the stub can be copied without using too much disk space. For example:

```
03/17/99  08:44a    16,384 pdfinitex.exe
03/17/99  08:44a    16,384 pdflatex.exe
03/17/99  08:44a   389,120 pdftex.dll
03/17/99  08:44a    16,384 pdftex.exe
03/17/99  08:44a    16,384 pdfvirtex.exe
```

There are four stubs linked to the same DLL. Should you create a new format file called `frpdflatex.fmt`, for example, you only need to copy `pdftex.exe` to `frpdflatex.exe` and the new format file will be loaded automatically by calling the new command, which has a very small footprint on disk.

There are other potential advantages:

1. upgrading to a new version of pdfTEX could be done by only upgrading `pdftex.dll`;
2. clever TEX shells could drive TEX engines directly by talking to the DLL and not use the command line.

Accessing files on the network. Under Windows, you can access files shared on the network by using UNC names. UNC refers to *Universal Naming Code*, a syntax introduced by Microsoft to refer to shared resources — files or devices — available through the network. The `kpathsea` library is thus made aware of UNC names, means you can make

[7] Shortcuts are not file links but rather redirections, available only through the Windows shell environment, not from the command line.

`$TEXMF` point to `\\TeXServer\TeXmf` or ask `dvips` to print on `\\TeXServer\printer`.

The registry. Under Windows, every program accesses the registry to retrieve its parameters and all required information. The registry is a database, shared across the network which encompasses the environment.

So the question arises: should the port of Web2C to Windows use the registry? The answer is no. The main reason is that it is not recommended that end-users modify the registry by hand — there is too much potential danger for their system. So storing the configuration into the registry would prevent users to easily change the way their tools behave. Temporarily setting environment variables is a quick way to modify `kpathsea` behavior and a very useful feature which it would have been unwise to remove. Users can fiddle with their configuration parameters exactly in the same way they would under UNIX: by either editing the `texmf.cnf` file or overriding parameters in the environment. It is always a matter of getting the best of both worlds.

Previewer

Motivation. It was not at first my intention to devise one. DVI format is not a *modern* format anymore. Even if it fulfills everybody's needs, it does not mean it will last. The new pdfTEX extension has demonstrated that DVI is not mandatory for a TEX system. Thus, devising a previewer for Win32 is not a simple task. Spending lots of time on a tool that might turn quickly into something obsolete is not very appealing.

But no TEX distribution would be complete without a DVI previewer: many TEX users stick to the good old (plain- or LATEX-generated) DVI format before any kind of PostScript conversion.

We still might argue that Ghostscript provides an accurate view of what will be printed, but the process of TEX → `dvips` → Ghostscript is somewhat slow and heavy for many documents.

So I ended up in looking at the xdvi source code. As I had no previous experience of Win32 *graphics* programming nor of X-Window *graphics* programming, so this was another reason for doing the previewer.

Porting *graphics* application. If we omit the interface, xdvi uses only a few primitives from X-Window: it only needs to draw bitmaps for glyphs and rectangles for rules. So the decision was made to adapt to Win32 everything that could be (the page reading and drawing mechanism, for example) and to rewrite the user interface part.

All but two of the C source files have been patched to compile under Win32 and the missing graphic primitives have been added. As well, a new user interface has been devised following the samples provided by Microsoft with their Win32 System Development Kit.

Some issues have been raised — and solved! — by the redisplay mechanism. The main problem with the redisplay was where it should happen: in memory or directly onto the display surface? The former was easier but had one major drawback: at a scale factor of 1 and 600dpi, an A4 color page would be huge (about 34Mb). So, on a second try, the redisplay was changed to draw directly onto the screen. In fact, both solutions are still in the source code but only one is activated.

This is also the same reason for Windvi not displaying PostScript inclusions at a scale factor of 1: Ghostscript is told to allocate the whole page because it can be asked to display raw PostScript code. So this time, Ghostscript would require the 34Mb page. It does work under Windows NT — albeit very slowly — but it is too heavy for Windows 9x.

This was also the opportunity to fully understand where Windows 9x and Windows NT are different. They share the same API but they behave in very different ways. For example, the first data structures I built and that used to work under Windows NT assigned one bitmap handle per glyph used in the DVI file. It was even pretty fast but the same program running under Windows 9x was slow and eventually crashed. Looking at the resources, all the graphics resources were used. How to explain such different behavior? In fact, the Windows 9x GDI — the kernel part that implements graphics services — allocates all the graphic objects in a few 64K stacks. The one dedicated to bitmap headers was quickly filled in when the DVI file was using even a low number of fonts.

Features. The features of Windvi are almost the same as those of xdvi. I have tried to mimic the behavior of xdvi whenever possible and, at the same time, to add Windows behavior via *status bars*, *tool bars* and *tooltips*. The most important features of Windvi include:

- monochrome or grey-scale bitmaps (anti-aliasing) for fonts
- easy navigation through the DVI file
  - page by page
  - with different increments (by 5 or 10 pages at a time)
  - go to home, end, or any page within the document

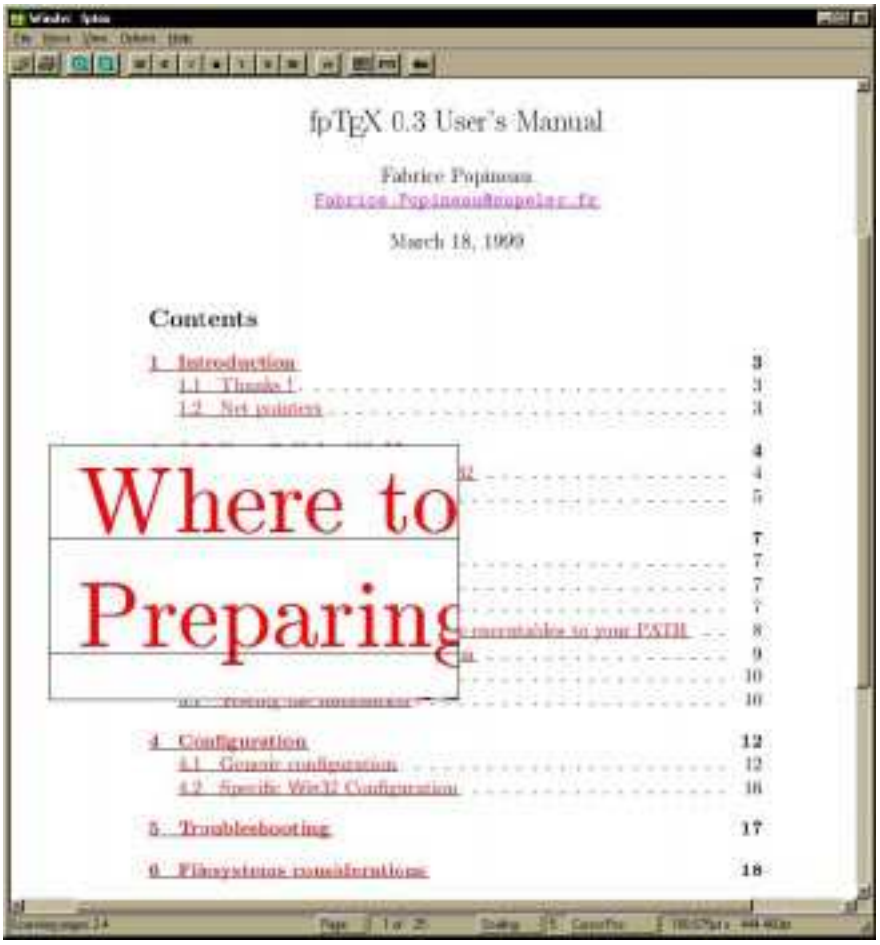


Figure 1: Windvi featuring magnifying glass and Hyper-TEX links

- different shrink factors to zoom page in and out
- magnifying glass to show the page at the pixel level
- compatible with xdvi keystrokes
- use of `.vf` fonts
- display `.pk` and `.gf` font files
- automatic generation of missing `.pk` files, even for Type 1 fonts
- tracking DVI file changes and automatic reopening
- understanding of Ω extended DVI files
- drag-and-drop file from the Windows shell explorer
- external commands through `\specials`
- color support (*à la* `dvips`)
- real-time logging of background font generation
- visualization of PostScript inclusions
- support of Hyper-TEX specials
- printing.

The main features not found in xdvi are color support and printing. The latter was again the opportunity to test different behaviors between Windows 9x and Windows NT — quite painful to debug. In fact, printing is something not at all easily done because there are many ways to handle it:

1. Print through the generic Windows printer driver, but PostScript specials will not be printed.
2. Ask `dvips` to convert the `.dvi` file to PostScript, and then either send the file directly to the printer, if it can handle it, or else call Ghostscript to do the job.
3. Build a bitmap with the page, PostScript specials included, and do *banding* (because the page would be huge) and send the bitmap to the printer.

Currently the first and third options are implemented but the third one uses lots of Windows 9x resources.

Last, the fact that Windvi is quite close to a port of xdvi was rewarding when it came to implementing the Hyper-TEX feature, which relies on the use of the `libwww` library, maintained by the W3C consortium. Fortunately, this library is available for UNIX *and* Windows. The net result was that adding this Hyper-TEX feature to Windvi took only a couple of hours to have a first workable result that allows navigation inside the document and referencing external programs. However, it turned out that, under Windows, this library is not mandatory at all because the shell can be called to open URLs, so it is not needed anymore.

Installation

Packaging TEX. Maintaining a `texmf` tree is a job that is very well done by Thomas Esser for teTEX and by Sebastian Rahtz for the TEX-Live CD. Given such a tree, I wanted to find a way to *automatically* group files by packages.

As has been pointed out in electronic discussion lists,[8] there is a lack of a standard procedure to install TEX packages. So there is no way to get a *source* `texmf` tree and *build* it, logging where every single file has been installed. So I wrote a few Perl scripts to reverse-engineer the build process, keeping the following goals in mind:

- Have three levels of completion: *basic*, *recommended* and *full*; given a package, these levels are guessed by heuristics from the `lists` files, devised by Sebastian Rahtz and to be found on the TEX-Live CD.
- Build a two-level structure targetted for InstallShield us (see next section); this structure is based on *components* (e.g. `latex`, `omega`, ...) and *subcomponents* (e.g. `latex\graphics`).
- Group files as much as possible; for example, to group fonts files, style files, source and documentation files for one package. This was done by implementing some kind of rule-based system in Perl, along with some other ad-hoc rules.
- Give a description for each sub-component; this was done using the Web description of TEX packages assembled by Graham Williams.

The result is not perfect, especially for the TEX-Live CD, with its huge number of packages. Notably, the automatic detection of descriptions is flaky — some of them being false, but this is harmless — and the recommended installation installs far too many packages, which means that the levels attributed to packages have been underestimated.

[8] See the dedicated mailing list on `tug.org`.

The installer. There is a product dedicated to building installers for Windows that is widely known and used in the Windows world, called InstallShield. Given a tree structure of *components* to which are associated *file groups* and a few *setup schemes*, it will build a nice looking installer.

But things are not so easy when it comes to installing a huge distribution. While InstallShield handles many common installation cases well, TEX is quite special because of the large number of files. This provides the opportunity to experiment with bugs and any limitations in InstallShield. Even though it is being used for the current release of fpTEX and the TEX-Live CD, it will probably be abandoned and replaced by a dedicated installer.

All in all, even if the installer is not perfect or flexible enough, it is useful enough to install from the TEX-Live 4 CD. The latest version of the installer used on fpTEX even makes it possible to add packages on top of an existing installation. And finally, it has been useful to use InstallShield to sort out all the problems related to installation; even if it is not used for future versions, the specifications are still there.

Windows integration. Most environments dedicated to TEX in one way or another will support fpTEX. Amongst them, we can cite WinEdt, 4TEX and XEmacs.

Configuration

Assuming a *recommended* installation, there is little to configure. But, as pointed out in the introduction, Windows users expect dialog boxes not text files to be edited. This has lead me to devise a dialog box-based tool targetted at fpTEX configuration. The `texconfig.exe` tool allows the user to access most of the configuration files in a point-and-click way.

Moreover, the standard Windows menus are provided with shortcuts to command-line tools (to rebuild formats or file database) and to local web pages (to access the documentation).

Future work

Some of the above-mentioned components will be enhanced in the near future.

Previewer. Even if the DVI format is old nowadays, many people are still using it so I will enhance Windvi in the following ways:

- Type 1 and TTF font support
- other graphics files format support
- graphical transformations for glyphs and rules under Windows NT

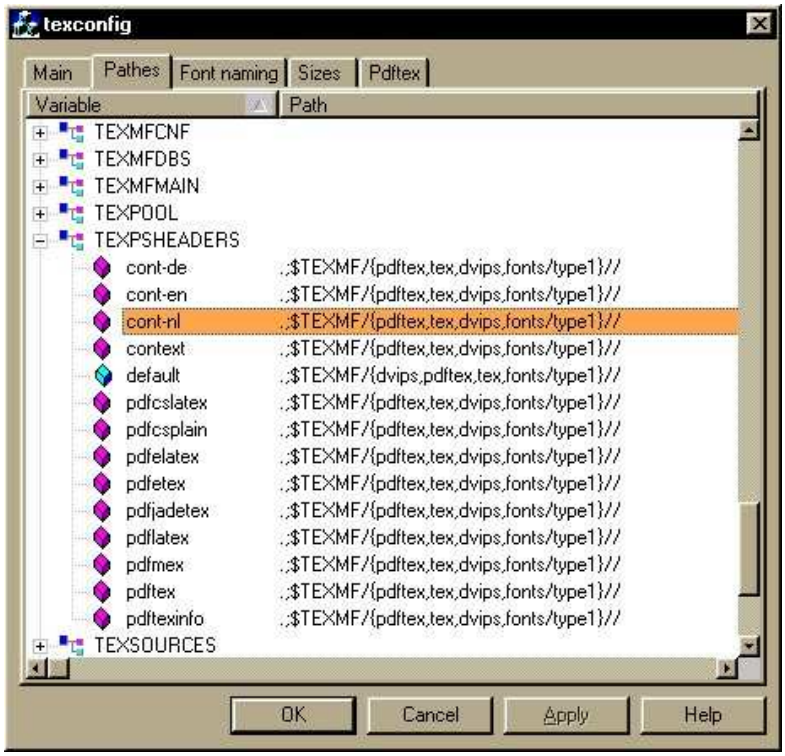


Figure 2: `texconfig.exe` tool editing pdfTEX related paths

- two-page spread mode
- forward and inverse search; that is, from the editor to the DVI file or from the DVI file to the editor
- cut-and-paste to other applications

Installation. The dedicated installer is being worked on. It will handle installations of both fpTEX and TEX-Live. The fpTEX files will be distributed as `.zip` files and if they are not present, the installer will try to download them from the net.

Configuration. The `texconfig.exe` program is not yet available. Its interface needs to be discussed because it is difficult to be simple and powerful at the same time. Many users want to tweak the configuration whereas, to make this thing really simple, we should hide most of the configuration parameters.

Availability

The whole package is available from CTAN, in the `systems/win32/fptex` directory. More information is available from `www.ese-metz.fr/~popineau/fptex`, the home of fpTEX. The TEX Users Group is kindly hosting a dedicated mailing-list, `fptex@tug.org`, to which you can subscribe by sending a request to `majordomo@tug.org`.

Acknowledgements

All this work relies heavily on the work done by Karl Berry, Thomas Esser, Sebastian Rahtz and Olaf Weber on the UNIX distributions Web2C, teTEX and TEX-Live.

Obviously, the numerous authors of all the packages present in fpTEX — programs or TEX style files — are thanked too for having shared their work.

References

Berry, Karl and O. Weber. “The Web2C distribution of TEX”. `http://tug.org/web2c`, 1999.

Esser, Thomas. “The teTEX distribution of TEX”. `http://tug.org/pub/teTeX`, 1999.

Gurari, Eitan M. “A demonstration of TEX4ht”. `http://www.cis.ohio-state.edu/~gurari/tug97/tug97-h.html`, 1997.

Popineau, F. “Rapidité et souplesse avec le moteur Web2c-7”. *Cahiers GUTenberg* **26**, 96–108, 1997.

Popineau, F. “Windvi User’s Manual”. *MAPS* **20**, 146 – 149, 1998.

van Dobbelsteen, G. “DVIview: A new previewer”. *MAPS* **20**, 120–124, 1998.

Managing TEX Software Development Projects

Jeffrey McArthur
ATLIS Publishing Services
8728 Colesville Road
Silver Spring, MD 29010
jmcarth@atlis.com

Abstract

During the past few years, many articles and books have been written about managing software development projects. Software development projects using TEX require some special attention. This presentation looks at the entire software development life cycle as it applies to TEX and the following issues in particular: requirements specification, design specification, coding standards, code review checklist.

Why use software development management techniques?

Writing good, versatile and well-documented code should be the goal of anyone developing macros in TEX. Unfortunately, even the macros that are included in the standard distribution of TEX fail that standard. Books typeset using TEX often have special coding scattered throughout the source to make the book layout better. The macros used, however, do not always work the way the documentation says. Book typesetting specifications are often incomplete and ambiguous or refer to the style of some other book without any detailed information on fonts, page size, and so on. Specifications can, and usually do, change and mutate during the production process. Poorly documented and bug-ridden macros make managing the process a nightmare.

Database publishing pushes the process to its limits. It is one thing to produce a book once, or even once a year; it is another thing to produce the same book each and every month, in a scenario in which the data changes on a daily basis and is extracted out of a database only for composition. Database publishing does not allow the luxury of scattering special code throughout the sources. The production process is often automated and the TEX typesetting macros must be written to take care of all contingencies because the input data file format cannot be adjusted to make the book layout better.

One solution is to use software development management techniques. This paper is an attempt to define a template or checklist suitable for managing a software development project that uses TEX as one of its primary programming languages.

Requirements specification

The first step in any software development project should be to create a document known as a requirements specification. This document defines the project in very broad terms. A copy of the requirements specification should be given to everyone involved in the project; any time a requirement changes, everyone involved should be informed of the change by receiving an updated requirements specification document. The requirements specification should include:

Description. I am amazed at how difficult it is to describe some projects. If it is not possible to write down a simple paragraph that gives a valid description of the project, that project is not well defined and has a high probability of getting out of control.

Project goal. Once the project is described a goal or goals can be defined: "If you don't know where you are going, then how can you know when you get there?" Defining the project goal can force important issues to surface at the start of the project. A written project goal provides a means to objectively measure the success or failure of a project.

Needless to say, the project goal should be documented and understood by all members of the software development team as well as the project manager. Failure to have a documented goal will lead to feature creep and project drift.

Overview of problems to be solved. Describe the fundamental problems that need to be solved. Converting the existing data into a usable format may be a real challenge. Word processing files,

poorly organized databases with little or no documentation, and spreadsheets can, and often are, the only available format for the data. All must be converted into a format suitable for typesetting with TEX.

Tasks/Functions. Specify the tasks or functions the macros perform. For example, if there are any extracted indexes or page cross references they should be defined.

Current mode of operation. It is useful to know the current mode of operation. This avoids the problem of creating a solution that cannot be easily integrated into the working environment of the user. For example, if the user is a hard-core plain TEX user, providing a set of macros that only work under LATEX would not be an appropriate solution.

Communications. Does the user expect the data back in some other format? One rather large project that I was involved in required converting data from a proprietary typesetting format into SGML, typesetting a book, and producing an electronic version of the data on CD. We finished the book and the CD and figured we were done for the quarter when the client called up and asked us about the "mag-tape version"? Our marketing department had forgotten to tell us that we had to create yet another deliverable. The data was to be delivered on an IBM formatted 6250 BPI magtape with a specific tape label. The requirements document should have specified that deliverable. Unfortunately it did not, and we had to scramble to pull together the resources to complete the project.

If the data undergoes any conversion processes it is important to specify the life cycle of the data and its changes. This means that the date (and possibly the time) when the data is frozen for production should be specified. If the data is modifed for typographic reasons during the production process, it must be specified if the original data is to be updated to match the typographic changes. For example, if the data is in SGML or XML it is common to add processing instructions to the data to help with the typography. It is sometimes necessary to make structural changes in tables, particularly CALS SGML tables,[1] to make them typeset properly. Changes of this type require complex changes to the source. If the data must be returned in a format that could be used again, the returned data must include the structural changes to the table elements.

Another example involves the creation of a printed book and an electronic product, e.g. CD-ROM. Typographic changes due to typesetting the book may need to be reflected in the CD. Failure to document this requirement can lead to serious problems if the products get out of sync with each other.

Ease of use. Specify how experienced with TEX the user of the macros should be. It is all too easy to create macros that only an expert can use. Occasionally, however, it is appropriate to create complex macros. The key is to document the expertise needed, since the level of experience of the user also defines the amount of detail that the documentation needs.

Hardware/Software. Specify the hardware platform(s), operating system(s), and implementation(s) of TEX that the macros will run on. This is important if you are using some of the modified implementations of TEX like ε-TEX, Omega, or PDFTEX.

Some implementations of TEX are compiled for a particular memory size. Other implementations are configurable. In either case, the minimum and recommended TEX memory size should be specified. This will let the user know if they have to use a larger version of TEX or reconfigure their existing version.

TEX has a very small memory footprint by today's standards. Unfortunately some of our users have just about everything running on the PC at one time. One of our users has the following programs running at all times: Excel, Word, Outlook, Internet Explorer, and several other proprietary pieces of software. "Bloatware" software packages can use up tremendous amounts of local disk space and memory. Attempt to define the possible interactions that might occur if other software packages are running. Define the amount of both network and local hard disk space required.

[1] Continuous Acquisition and Life-Cycle Support (formerly Computer-aided Acquisition and Logistics Support) (CALS) is a Department of Defense (DoD) strategy for achieving effective creation, exchange, and use of digital data for weapon systems and equipment. MIL-PRF-28001C is the military standard for CALS markup requirements. A soft copy of this is available at:

`http://www-cals.itsi.disa.mil/core/standards/28001C.pdf`

Because CALS tables are designed to support the entire DoD they are very complex and difficult to use.

Quality. Outline in broad terms the rules for word, paragraph, column, and page breaking.

Performance. TEX is a high-performance typesetting system. Usually there is no need to worry about performance. However, the generation of PDF pages on demand by a web server can become a performance issue. List all performance criteria.

Compatibility and migration. Specify if the macros are based on plain or LATEX or something else. Specify if the macros have to be compatible with any other macro package. If the macros are a replacement/upgrade of an existing package specify what amount of re-learning will be required of users as they migrate their data to the new package.

International. Specify the need to support running and/or continued heads that may include support for ® and ™ as well as accented characters. Specify how ® and ™ are to be typeset: superscript or not, serif or sans-serif or font dependent.

Itemize the languages to be supported. Each language requires it own hyphenation dictionary. The encoding of the input data should be defined. Determine if the data uses the Latin-1 encoding or some other format, e.g. UTF-8.

There is a lot of data with accented characters that uses MS-DOS code page 437 or 850.[2] This data is not compatible with the TEX standard encoding or 8r, used with most PostScript fonts. The way the data is encoded should be documented.

Service and support. Itemize the level of service and support. The days and hours when support is available should be listed in detail.

Pricing/Licensing. Define the ownership of the macros. Specify the method by which the source code will be provided and if the source code can be de-commented.

Design specification

Creating the design specification document can be done in parallel with creation of the requirements specification document, but it should not precede it. It is important to understand what is required prior to defining the design of the pages.

Publishers often provide design specifications, but publisher-supplied design specifications are often incomplete, vague, and ambiguous. Even when the publisher provides such information, a document should be created that defines all the details required by the project.

Documenting the specifications also gives the typesetter a mechanism to generate additional revenue when the publisher makes changes at the last minute.

Description. Give a detailed description of the typeset pages.

Output format. Define the output medium. Resin-coated paper and film are still used as well as electronic formats. The design specification document should unambiguously define the format of electronic files. There are many possible standard electronic formats: PostScript, PDF, or separate EPS files. If the deliverable is in PostScript, identify which PostScript level 1, 2 or 3 is required, and if the files are to be DSC-compliant (Document Structure Convention). Specify how many pages will be in each output file.[3]

If the final deliverable to the printer is to be electronic files, it is important to establish a dialogue early in the process with the printer's technical staff in order to determine the specific file formats needed.

Covers/Spines. State if book covers and/or spines are part of the deliverable.

Page size. Define the physical size of the page. In PostScript this is also the bounding box. If there are crop marks, the size of the physical page will need to be larger than the size of the trimmed page. The size of the trimmed page should also be defined.

PostScript and imposition software may have additional requirements. The bounding box of the printed page may or may not need to include the crop marks, depending on the needs of the imposition software.

Crop marks. Specify if crop marks are needed and where they are to be located. Today, many web press printers want crop marks, but they must

[2] The term "code page" refers to the keyboard and display encoding. When MS-DOS was developed there were no accepted standards for the layout of accented characters. Code page 437 was the default layout for the version of MS-DOS that what shipped to the United States, and code page 850 was the default layout for Wester Europe.

[3] Some imagesetters limit the number of pages they can process in a single file. Often long runs must be broken into ten or twenty page batches. When this happens, the design specification document should also define the file naming convention for the multiple pieces.

be located one-eighth of an inch outside the page. Specify the location of the crop marks and what they should look like.

Color separations and registration marks. Define the number of color separations and what information is printed on each separation. Registration marks should also be specified. The use of spot color or highlighting also needs to be defined.

Screens. Specify if there are any screens on the pages or if the pages are to be set on colored paper. Also define the amount that the screens must extend beyond the trim size.

Bleed tabs. Define the number, size, and placement of any bleed tabs as well as the amount that the tab should "bleed" beyond the trim size.

Makeup. Define the number of columns per page for each section of the book. The rules for starting a new column, new page, and new right-hand page should all be specified.

Fonts. Define what fonts are to be used. If the output is PostScript or PDF, the document should also specify if the fonts are to be embedded in the document. Also specify if embedded fonts must be the entire font or if the font can be a subset of only the characters used by the document.

Running heads. Define the running heads. The rules for any alpha-omega or dictionary heads should be specified. The document should also define if math, accented characters, or other complex textual material can occur in the running heads since these may require particular attention.

Even if there is initial agreement that there will be no math in the running heads, this may change later in the process. Any changes to design specification must be agreed to by all parties.

Continuation heads. Define the number and type of continuation heads. As with running heads, the document should define if math, accented characters, or other complex textual material can occur in continuation heads.

Sorting. If the data is to be sorted, the rules for sorting must be defined. Particular attention should be made to rules for ignoring leading articles such as "a", "an", and "the", casing and punctuation. The sorting order for names can be particularly complex and must be defined.

Sorting languages like Chinese can be particularly challenging since it can be sorted in either radical then stroke order or stroke then radical order. The order is dependent upon the publisher's preference. Dealing with sorting and punctuation in Chinese is particularly painful. Japanese has even more complicated sorting requirements. All of these need to be specified if the sorting involves Chinese, Japanese, or Korean.

Cross-references and hypertext links. Define the types of cross-references. The document should define if they are simple references, or actual page cross-references or even hypertext links. The formatting style for hypertext links should be defined.

Extracted indexes. Define any extracted indexes. The sorting order for each extracted index must be clearly described and all exceptions noted.

Graphics and line art. Define the parameters for line art. If there are specific limits on the art size or shape they must be documented. The acceptable formats for the artwork should be specified. If the artwork is to be scanned, the scanning resolution should be specified as well as the delivery format. If there is an approval process associated with the artwork[4] it should be clearly spelled out.

Hyphenation. Define the rules for hyphenation. Any multi-lingual document requires special attention to defining the hyphenation rules.

Widows and orphans. The rules for breaking paragraphs, columns, and pages should be defined in detail. Special attention should be devoted to pages that run short or long.

Additional breaking and grouping logic. Describe any logic that may be required for grouping. For example, it may be undesirable to break address listings except at specific places. For example: the city, state or province, and postal code should sit as a block, except when they will not fit, and then it should break following the city. The rules for addresses outside of the United States and Canada should be defined in detail.

Blank pages. The direct-to-plate technology does not come without a price: imposition software demands consistency. Often it is necessary for the final deliverable electronic pages to include blanks. When a section of a book is defined to start on a new right-hand page, the typesetting macros may need to output a blank page instead of just incrementing the folio. Bleed tabs, crop marks, and screens complicate the process.

[4] Advertisements in books are generally artwork that must be approved prior to printing in the finished book.

Front and back matter. Cover pages, copyright pages, and dedications are often created using WYSIWYG software. If the book is to be printed using direct-to-plate technology these pages may need to be integrated into the electronic files for the body of the book. How these pages are to be delivered and who will be responsible for the integration must be documented.

Typesetting deliverables. Define how the finished pages are to be delivered. In the case of film or photographic paper, include the shipping address and how the package is to be shipped. If the deliverable is electronic, specify how the data is to be transferred or shipped. Define the acceptable media: floppy, zip-disk, CD-ROM. If the files are to be electronically transmitted, list the email address or the ftp site to which the data is to be sent. If the data is to be transferred through a firewall, the security measures should be specified.

Features/Enhancements. Define possible future features or enhancements that could be made to the typesetting macros.

Exit conditions. The printer has the final word on the design specification document. The location of crop marks, bleed tabs, registration marks, and screens may need to be adjusted to meet the demands made by the printing press. Sample pages should be sent to the printer for their approval as soon as possible in the process.

Metrics

Dr. W. Edward Deming, a well-known author on management techniques and practices, introduced various quality control methods into management practice. He emphasized that you cannot manage what cannot be measured — otherwise you have no idea if what you are doing helps, harms or has no effect. He also introduced a number of statistical techniques for measuring product quality, as well as procedures for measuring improvement.

Therefore, as part of managing the development process, it is important to create an objective measure of the quality of TEX macros. As a programming language TEX has some unusual features such as the ability to change the category code of characters. This makes it difficult to create accurate metrics. The solution is to enforce coding standards.

Metrics are a complex and controversial subject that requires more than just a few paragraphs. I plan on writing a detailed paper in the near future, to cover the topic of metrics and TEX.

Coding standards

There is very little literature about coding standards for TEX. ProTeX and docstrip are tools that are supposed to aid in documentation and code generation, but neither assures consistency in coding style.

Introduction. This is an attempt to introduce a formalized set of standards for software development using TEX. During the past nine years I have managed nearly twenty man-years of extensive development in TEX and this paper is based upon that hard-earned experience. My focus has been exclusively on plain TEX, although most of these standards can be applied to LATEX.

Scope. This set of coding standards and conventions for coding and commenting TEX macros will help ensure consistency and maintainability. These standards were created not only for newly developed code; any maintenance change to existing code should attempt to bring that code into conformance with at least the commenting standards.

Purpose. Coding standards provide a framework for developing code that is both internally and externally consistent. The framework should provide the support necessary to allow the programmer to concentrate on creating the best implementation of the code.

Deviations from industry standards. *The TEXbook* defines a coding and commenting standard by its numerous examples. The coding style used in *The TEXbook* puts multiple statements per line and uses trailing `\fi`. This style makes it difficult to follow the logic of the code.

Instead of continuing to emulate the standard defined in *The TEXbook*, this is an attempt to define a new standard that treats TEX macros like any other software development language.[5]

Some will argue that the coding styles in printed books such as *The TEXbook* and *TEX: The Program* are used to save space in the printed product. Paper-saving economies that may have been exercised for budgetary reasons should not have a long-term bearing on standards particularly if the result is compromised clarity. TEX is about fine typesetting. Listings of code should be held to the same high standard as typesetting text.

There are tools to help with coding. The editor I use has a mode that is supposed to assist with the formatting of TEX. I find it more trouble than it

[5] McConnell (1993) is an excellent reference and is the basis of much of this standard.

is worth, particularly since I rarely work with the standard `\catcode` settings.

This points out the problems with tools like funnelweb and ProTeX: they place restrictions on the type of data the macros can be used with. If your input file is in SGML or XML, there is absolutely no reason to preprocess that file before using it with TEX. It is relatively simple to typeset SGML or XML data using TEX.[6]

Working directories. The TDS, TEX Directory Structure, is designed to stabilize the organization of TEX-related software packages. Unfortunately the TDS does not work in an environment where there are multiple projects that use TEX as an embedded typesetting engine or in an environment where there are multiple projects where TEX is only one of a number of tools used. It is preferable to keep all the macros under development in close proximity to the rest of the project and not part of the TDS tree.

The directory structure in use at ATLIS Publishing Services is quite different from the TDS. The top-level structure is based on accounts. Ideally, each account is broken into projects. Under each project specify the following directories:

- `doc` for Documentation and correspondence
- `help` for help files
- `version` for version control files
- `alpha` for distribution files that are part of the current alpha release
- `beta` for distribution files that are part of the current beta release
- `release` for distribution files that are part of the current production release
- `tex` for TEX files
- `sgml` for SGML, DTD files and such
- `lex` for Lex files
- `Delphi3` for Delphi 3 files
- `Delphi4` for Delphi 4 files
- and so on for other project-specific files

Format files and such are built and copied to the `alpha` directory where they are tested. When the macros have passed regression testing, the alpha files are moved into the `beta` directory as part of a 'beta' release. When the 'beta' release has been tested by the end users and accepted, the files are copied into the 'release' directory.

[6] The website `www.greenbook.net/free.asp` allows viewing of thousands of SGML documents that were typeset using TEX without the use of any preprocessor. XML is a subset of SGML.

File naming conventions. Below is a table showing the proposed file-naming conventions for macro and font files:

Prefix	Extension	Description
	`.tex`	source file
	`.sty`	macro include file
	`.fnt`	font include file
	`.dat`	data file
Tst	`.tex`	unit test file

Source file. TEX uses `.tex` as the default extension for input files. This extension should only be used for files that can be used on the command line for TEX. That is, any file which is to be run by itself through TEX or any file that can create a format file. If the file will only run with a particular format file, e.g. LATEX files, then the extension should not be `.tex`. It should be possible to determine the purpose of the file and how to process the file from its file name. Using the `.tex` extension for files that require the LATEX format is counter-intuitive because the extension implies that the file will work with TEX and does not imply that a format file is required.

Macro include file. Style, or `.sty` files, contain macro definitions. A `.sty` file should not produce any output to the `.dvi` file.

Font file. TEX provides a tremendous amount of power in its use of the `\font` primitive. Unlike some desktop packages, TEX allows complete control over how fonts are loaded and how they are used. One of the goals of good macro design is to separate form from function. That is, the data should be tagged as to its purpose and not as to how it looks. This philosophy should also be reflected in the way fonts are loaded. `\font` statements should not be mixed with macros. Fonts should be loaded as part of a separate file (or files). This makes it easier to change the fonts used to typeset a document. The document itself should not reference any font by anything but a generic name. The New Font Selection Scheme (NFSS) follows this same principle.

To promote this methodology the `.fnt` files contain all the `\font` statements. This has some significant advantages over the standard LATEX method of specifying fonts. LATEX allows the user to specify the main point size of the document in the `\documentclass` statement. Changing the main point size of the document requires the main data file be modified. It is better to separate all references to fonts and font sizes from the document.

File names. Having a portable file name versus with an understandable file name is the main

issue in choosing a standard for file names. "8 + 3" file names are more portable than long file names but it is difficult to use meaningful file names with only eight letters. All users should be polled to ensure that they can process the long file names, however. To avoid any possible problems, all file names should be limited to strictly alphabetical characters.

Coding conventions.

White space or blank lines. Blank lines should be used to show the organization of the file. Files with no blank lines are difficult to read and understand.

Dividing lines. By default TEX uses the percent sign, %, as the comment character; it also makes a good section divider. I recommend using a line of percent signs to separate sections of text. It is possible to mark off major sections by using double lines of percents. Minor sections can be delimited by using half lines or quarter lines of percent signs.[7]

Version control. Keeping track of revisions, releases, and versions of software is important to all successful software development projects. Integrating TEX with a version control system is relatively simple. There are numerous version control systems, each with their own features and syntax. Each set of TEX macros should announce to the user what version of the macros they are running. The following code fragment shows one method of passing the version information to a TEX macro:

```
% First for the revision level
\def\DefStyleVersion'#1'{%
    \gdef\StyleVersion{1.#1}}

% ***keyword-flag*** '%v (%d %t)'
\DefStyleVersion'2 (5-Aug-98 2:42:04)'
```

The version number should be announced to the user by using `\everyjob`.

General coding conventions. The Fundamental Theorem of Formatting is that good visual layout shows the logical structure of a program (McConnell, 1993:403). TEX macros should use a layout style that:

- accurately represents the logical structure of the code
- consistently represents the logical structure of the code
- improves readability
- withstands modifications as the code is maintained

Modularity. Macros should be written in a modular fashion. For example, the macro should not call out fonts by their point size but rather by their usage. So, instead of using `\tenrm` the macro package should reference something like `\NormalRoman`. This allows the fonts to be replaced conditionally depending on context.

In almost every case, over the past nine years of software development projects using TEX, the font set had to be changed at some time during the project. In many cases different outputs were created using different sets of fonts.

The same input file can be used to create dramatically different output formats. One project entailed typesetting a directory of telephone and fax numbers. The input composition file was approximately 60 MB in size. From the single composition file two different directories were created. The first, much larger, included both the phone and fax numbers. The second, much smaller, contained listings only with fax numbers. The TEX macros were written to programatically suppress listings in the directory if they did not have a fax number.

Macro coding conventions. TEX macros should be self-documenting. In other words, the code should be commented in such a way as to make it easy for the casual reader to understand what the macros are doing, no matter how complicated the actual logic is. Each macro should have a preamble comment. The preamble should define what the macro does. If the macro takes parameters, each parameter should be documented as to what it is and what its expected value should be. If the macros take optional parameters then those optional parameters should also be documented.

Indentation. To accurately and consistently represent the logical structure of the code, macros should be formatted using a block indentation style.

"if" coding conventions. All if–else–fi testing should show the logical block structure of the code. Below is an example of code that does not show the logical structure:

```
\def\strut{\relax%
        \ifmmode\copy\strutbox%
    \else\unhcopy\strutbox\fi}
```

The logical structure is much easier to understand using the following formatting style:

[7] Some editors allow the user to specify how many "repeats" of a character to use, a function which facilitates insertion of such dividing lines.

```
\def\strut{%
    \relax%
    \ifmmode%
        \copy\strutbox%
    \else
        \unhcopy\strutbox%
    \fi%
}
```

"elseif" coding conventions. TEX does not define an `\elseif` primitive. Occasionally it is useful to do several sequential tests, such as testing a parameter to see if it matches some pattern. Because the tests are sequential, a strict indentation style would be difficult to read. In those cases a modified indentation style should be used:

```
\def\TestValue{%
    \ifx\next\ValA%
        \ProcessA%
    \else\ifx\next\ValB%
        \ProcessB%
    \else\ifx\next\ValC%
        \ProcessC%
    \else%
        \ProcessOther%
    \fi\fi\fi%
}
```

Specialized TEX coding conventions. The code necessary to change the value of the TEX escape character, \, in order to process verbatim text or produce auxiliary index files, is difficult to document. In cases like this, there should be an extensive preamble that documents the process.

User interface. TEX is a batch processing typesetting system and as such has a very rudimentary user interface. However, the user should be able to tell at a glance what version of the macros they are running. TEX provides an `\everyjob` facility that allows format files to show the user the information about the version of the macros.

As TEX processes pages it displays the folio. For large jobs, it may also be desirable to inform the user what part of the document TEX is processing. This can be done using `\write` or `\message` statements.

Code review checklist

Prior to doing a code review, the software should undergo a clean build, followed by an inspection of its design and coding.

Clean build process. Prior to the release, the software should be subject to a clean build and test. Doing a clean build ensures that all the files are properly checked into the version control system and that it would be possible to recreate the project from a backup of only the source code.

Back up everything. The first step in a clean build is to back up everything. This is important because, as part of the process, many files will be deleted.

Check everything into version control. Make sure that all source files are checked back into the version control system. This also ensures that version/revision numbers are incremented.

Delete the entire project. All source code files and format files are deleted from the system. Those who are worried about disaster recovery would start with a system with a completely clean disk and would require all the development software to be reinstalled.

Restore the source from version control. Restore all source files from the version control system.

Build the project. At this point the format files should be rebuilt. One of the most common problems is missing files. If a file is missing, it was not included in the version control system.

Regression testing. Testing to make sure that software has not taken a step backward and reintroduced bugs that have been fixed previously is called regression testing. Because the entire system has been rebuilt, it is important to check that nothing has been inadvertently changed.

Design and coding inspection. The clean build should find any files not included in the version control system. Every project should have a document that defines how it is to be built and this document should be updated to list any problems that appeared during the clean build.

Using the Design Specification Document, the code should be inspected to see if it is easy to determine if the code implements the design specification. Items that should be checked include:

- page size
- crop marks
- color separations and registration marks
- screens
- bleed tab
- page makeup
- fonts
- running heads
- continuation heads
- cross-references and links
- extracted indexes
- artwork
- additional hyphenation patterns

- widow and orphan logic
- blank page generation

Project plan and status reporting

Successful management of a software development project requires that a detailed plan be developed. The plan should be created using project management software. As a rule of thumb, all tasks should be between four and sixteen hours in length. If the estimated time for task is longer than sixteen hours, the task should be broken up into sub tasks.

A status report showing the state of the project should be created weekly. Because estimating the time for a software development project using TEX is difficult, it is important that the time for the development be tracked against the plan. Tracking the time provides feedback on the estimate, allowing the estimating skills of the project manager to improve.

Summary

The Software Engineering Institute, or SEI, has defined a Capability Maturity Model, or CMM, for the software development process.[8] Briefly, the CMM levels involved are:

1. initial: no formalized procedures
2. repeatable: basic project management
3. defined: process standardization
4. managed: quantitative management
5. optimizing: continuous process improvement

This paper is an attempt to move from level 2 to level 3. The process for managing TEX software development projects must be defined so that the process can be managed, level 4, and optimized, level 5.

CMM levels provide an objective measure of the quality of the management process. Better managed projects provide higher customer satisfaction and lower costs. The standards I am proposing will encourage macro re-use and improved documentation, both of which should result in improved efficiencies, cost-containment, and easier transfer of maintenance and support duties to individuals other than the original coder.

Acknowledgements

I would like to thank the reviewer for their time and patience in reviewing this article. In particular, the introduction to the section on metrics was vastly improved by their comments.

A special thanks to Melissa Colbert and Denise Marcus, who helped me prepare this paper for submission.

I would also like to thank Christina Thiele for the thankless work as editor.

Selected bibliography

Arthur, Lowell Jay. *Improving Software Quality: An Insider's Guide to* TQM. John Wiley and Sons, New York, 1993.

Constantine, Larry L. *Constantine on Peopleware*. PTR Prentice Hall, Englewood Cliffs, New Jersey, 1995.

Humphrey, Watts S. *Managing the Software Process*. Addison-Wesley, Reading, Massachusetts, 1990.

McConnell, Steve. *Code Complete: A Practical Handbook of Software Construction*. Microsoft Press, Redmond, Washington, 1993.

Whitten, Neal. *Managing Software Development Projects*. John Wiley and Sons, New York, 2nd ed., 1995.

Yourdon, Edward. *Decline and Fall of the American Programmer*. PTR Prentice Hall, Englewood Cliffs, New Jersey, 1993.

[8] SEI's website is: `http://www.sei.cmu.edu`; the CMM material begins on `/cmm/cmms/cmms.html`.

JAVA and TEX

Timothy Murphy
School of Mathematics
Trinity College Dublin
Ireland
tim@maths.tcd.ie

Abstract

TEX and METAFONT, translated into JAVA and compiled with TYA, a public-domain JIT compiler, run about 10 times more slowly than the same programs in C (without TYA, they are 20 times slower). A year ago, they ran 50 times more slowly. In a year's time, perhaps using Sun's new JIT compiler, it is reasonable to assume that the factor will be down to 2 or 3.

At that point — bearing in mind the speed with which the speed of computers is increasing — TEX-in-JAVA will be a perfectly plausible alternative to TEX-in-C; and then we shall have to weigh its lack of pace against the several advantages that JAVA has to offer, such as, portability, "netability", modularity, threads, and graphics.

Has TEX found its natural niche?

TEX has attained a complete monopoly of the mathematical market. (Are there still primitive people somewhere in the world speaking `eqn`?) And as mathematics continues its remorseless march to colonize new areas of knowledge, it carries TEX (like a disease) with it.

At the same time, it must be admitted that TEX has been less successful outside these areas than was hoped for, say 10 years ago. Of course that is not a disaster. According to Ken Thompson (creator of UNIX), "a program should do one thing, and do it well", and it may be that mathematical typesetting is the one thing that TEX does well, indeed superbly well. It would be foolish to risk this in pursuit of some universal rôle.

However, the cause is not necessarily lost. In the author's view, the solution does not lie in the addition of yet more features to TEX/LATEX — features which all too often satisfy the needs of the cognoscenti at the cost of complication for the newcomer — but rather in a more rigorous analysis of the TEX engine, and of the function and relation of its parts. It is the author's thesis that JAVA can provide the stimulus to set such an analysis in train.

It should be emphasised that the author is not suggesting — indeed, would be bitterly opposed to — the creation of yet another "near-TEX". The only threat that TEX faces in the medium term is fragmentation. All religions agree on one thing: that the greatest danger comes from within. Today, none of the schismatic versions of TEX (with the possible exception of PDFTEX, which denies the accusation of heresy) has a measurable share of the market; but if NTS, let us say, were to gain the allegiance of 25% of TEX users then the future of TEX — and NTS — would be in doubt. (For a more sympathetic view of NTS and other TEX extensions, see Knappen, 1995.)

The danger of such fragmentation can be seen clearly in the failure of literate programming' to fulfil the promise vested in it by Knuth (1992). The proliferation of innumerable *`WEB` programs (and one should include with these the LATEX `doc` system) — each of them doubtless superior in some aspect to Knuth's original — far from leading to widespread adoption of the literate programming paradigm, has stifled it almost to death.

The JAVATEX project

Although the principal aim of this talk is to demonstrate `DviPdf`, a TEX output driver written in JAVA, it may be more useful in this written version to say a little about the JAVATEX project of which `DviPdf` is part. This project has two main thrusts:

1. To translate the classic `.web` files (`tex.web`, `mf.web`, `tangle.web`, etc) into JAVA, using `web2java`, a straightforward modification of the standard `web2c` translator.
2. To develop output drivers — and other TEX support programs — in JAVA, using the standard Knuth/Levy `cweb`, modified (slightly) to output JAVA rather than C.

Why Java? JAVA has several advantages over C as a medium for TEX software.

Portability: In principle, a JAVA application — expressed as a number of communicating JAVA classes — should work unchanged on all platforms supporting JAVA, which means, in effect, under every OS.

Netability: JAVA was designed with the Internet in mind, and its adoption should allow TEX to be integrated more easily into the Web.

Modularity: JAVA is *object-oriented*, allowing classes to be shared by different programs, so that, for example, all drivers can share the same font manager and file server, and use the same DVI reader, And one can define an abstract generic driver, minimizing the size of actual drivers.

More speculatively, although TEX and METAFONT are large monolithic programs, they are actually written in a modular style — almost as though Knuth had JAVA in mind! — and it should be relatively simple to "hive off" font routines, for example, as a separate `TeXFont` class, without modifying the essential code in any way. Breaking up TEX (or METAFONT) in this way into a number of co-operating classes might mean that variations such as PDFTEX and METAPOST could be implemented as relatively small extensions of one or more of these classes.

Graphics: The standard graphical interface built into JAVA — but interpreted in the style of the platform in use — should mean that the same TEX viewer can function under UNIX, MS-DOS and Mac. And this interface would also offer a graphical alternative to the perhaps old-fashioned text-based interface traditional to TEX.

Threads: There are some advantages in running the different parts of a program as separate threads. For example, a font server can "sleep" until a font is requested; in an integrated system it may serve more than one program or even more than one user. By running TEX and friends as "threads", last-minute changes (as, for example, changing the sizes of arrays) can be implemented before the thread starts, and a program can pause while some intermediate task is performed, before resuming where it left off.

But JAVA is so slo ... ow? But that is part of its charm!

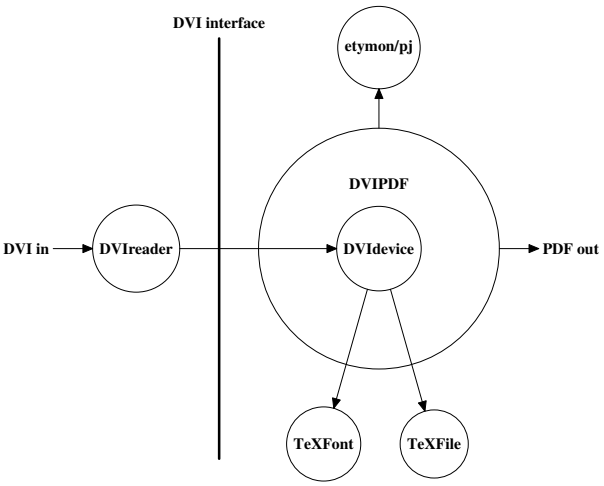


Figure 1: Anatomy of a driver

What is this world if, full of care
We have no time to stand and stare.

William Henry Davies (1870–1940)

At least things are getting better — three years ago, JAVA was 50 times as slow as C. Today, it is only five times as slow (with JIT compiler). Hopefully, this ratio will slowly approach a limiting value between two and three.

Sadly, Sun's long-awaited HotSpot compiler — now available (free of charge) on several platforms (Sun Microsystems, 1999) — failed signally to fulfil its promise that it would make JAVA applications run as fast as those in C++. It turns out to be little better than other JIT compilers.

A TEX output driver

Although we have chosen to illustrate our talk with DVIPDF — translating DVI input into PDF output — most of the code is common to all our TEX output drivers. The program is divided into seven parts (Figure 1):

The `DviReader`: This reads the DVI document, and translates the `DVI` commands into 'messages', as specified by

The `DVI` interface: This provides a "*cordon sanitaire*" between the `DviReader` and the driver proper.

The `DviDevice`: All output drivers share a great deal of functionality. For example, all treat fonts in much the same way. JAVA allows us to define a generic, or *abstract*, driver — `DviDevice` — containing this shared code. This abstract driver `implements DVI`; that is to say, it provides methods responding to the messages

sent by the `DviReader`, as specified by the DVI interface.

The `DviPdf` driver: Several of the methods provided by the generic driver `DviDevice` are empty; it is left to each concrete driver — such as `DviPdf` — which "`extends DviDevice`" to provide proper methods in these cases.

`TeXFont`: From TEX's point of view, all fonts look much the same. We express this by defining an abstract `TeXFont` class. Each font type — `.pk` fonts, Type 1 or Type 3 PostScript fonts, `.tfm` font descriptors, virtual fonts, etc — extends this class by adding its own particularities.

`TeXFile`: This is our "file manager", a highly simplified analogue of `kpathsea`. In effect, it uses JAVA's `Hashtable` class to construct a database of the `TEXMF` tree (or trees).[1]

The `PDF` classes: `PDF` — Adobe's anointed successor to PostScript — is object oriented, and thus particularly well-suited to JAVA.

Fortunately, there is an excellent library of JAVA classes — the `pj` library from Etymon Systems (1999), freely available with source — for reading and writing PDF files. Each kind of PDF object — font, page, etc — is represented by a `pj` class, with methods appropriate to that object.

In effect, the job of DVIPDF is simply to build up a PDF object; it can then be left to the `pj` classes to present that object to the world.

The DVI interface. The JAVA *interface* provides an exemplary tool for dissecting an application (i.e. a program) into independent parts which communicate according to the strict protocol laid down in the interface definition.

The DVI interface specifies 15 kinds of "messages". Any driver that `implements DVI` must provide 15 methods for responding to these messages. Since the definition of the interface is short and sweet, we give it in its entirety:

```
public interface DVI {

  void moveRight( int dh );
  void moveDown( int dv );
  void moveTo( int h, int v );

  void defineFont( int f, int checkSum,
    int scaledSize, int designSize,
    String area, String name );
  boolean setFont( int f );
  int setChar( int c );
  void putChar( int c );
  void setRule( int wd, int ht );
  void putRule( int wd, int ht );

  void special( String message );

  void bop( int count[] );
  void eop();

  void preamble( int numerator,
    int denominator, int mag,
    String comment );
  void postamble( int tallestPage,
    int widestPage, int maxStackDepth,
    int noOfPages );

  DataInputStream dviStream( int c );
}
```

All these methods (except the last) will be more or less self-explanatory to those familiar with the DVI format.

The first two "motion methods", `moveRight()` and `moveDown()`, describe relative motion, while the third, `moveTo()`, is absolute.

We note that while communication is primarily *from* the `DviReader` *towards* the "front end" of the driver, information can be passed back through the return value of the function or method. Thus `setChar()` returns the width of the character (which is all the `DviReader` needs to know about it), while `setFont()` returns `true` or `false` according as the font is *virtual* or *real*.

The last method, `dviStream()`, is the only one which is not immediately suggested by the DVI format. It is required to implement virtual fonts. A virtual character — that is, a character in a virtual font — consists of a fragment of DVI code, which must be integrated into the DVI document proper. In effect the input stream must be temporarily diverted to the sequence of DVI commands constituting that character.

The `DviReader` knows if the current font is virtual, from the return value of the last `setFont()`. If that is so then every character encountered until the next `setFont()` is virtual. After reading such a character, say character number c, the `DviReader` sends the `dviStream(c)` message to the device, which consults the appropriate font and points the reader to the new stream. (We use here the nice property of JAVA, that it can treat information in a file, and in a string, on the same footing.)

[1] Am I alone in finding `kpathsea` excessively complex? The bureaucracy of TDS (the TEX Directory Structure) seems to me entirely misplaced. Surely the computer was designed precisely to relieve us of such tedious (pun intended) tasks? Does it really matter if `.tfm`s and `.pk`s and `.sty`s find themselves in the same bed?

Virtual fonts and superfonts. A virtual font contains a number of *local* fonts. These are normally *real* fonts, but in principle they could themselves be virtual fonts.

This recursive potential of virtual fonts does not seem to have been exploited. It means in effect that the fonts in a TEX document form a tree, the leaves of which are real fonts, while the internal nodes are virtual fonts.

It is a natural step to connect the set of fonts by introducing a *superFont*, of which the top-level fonts — those actually named in the DVI document — are local fonts.

Recall that a virtual character is a fragment of DVI code. This suggests that we might regard the DVI document itself as a character — let us say character 0 — in the superFont.

It is amusing to take this conceit a little further. Different DVI documents could be characters 1, 2, 3, . . . , in the same superfont. Moreover, the superfont could itself be a local font in a super-superfont, which could itself . . . . A whole library of TEX documents might be organised in this way.

Tools

It is an essential feature of the JAVATEX project that all code in the package is written in Knuth/Levy `cweb` format, slightly modified (as described below) to output JAVA rather than C.

Thus the `DviPdf` driver is encoded in the files `DVI.w`, `DviDevice.w`, `TeXFile.w`, `TeXFont.w`, etc. (By convention, `cweb` source files carry the extension `.w`, to distinguish them from the classic PASCAL-based `WEB` files, which carry the extension `.web`.)

As mentioned earlier, the JAVATEX project also encompasses the translation of Knuth's classic `WEB` programs into JAVA, using `web2java`, a development — in some ways, a simplification — of `web2c`, the core program in the UNIXTEX implementation of TEX and its relations.

As an exercise, we base our `DviReader` on `dvitype.web`, which Knuth provided as a model for drivers. Thus `DviReader` is defined by a change file `DviReader.ch` to `dvitype.web`. The resulting PASCAL file `DviReader.p` is then translated into `DviReader.java` by `web2java`.

We end this note with a necessarily brief description of these basic JAVATEX tools.

Cweb for JAVA. Knuth's original `web` format was tied to PASCAL. Later Knuth and Levy developed `cweb` to provide output in C. Since JAVA is a dialect of C, `cweb` only requires minor modifications to output JAVA. These are contained in the change files `ctang-java.ch`, `cweav-java.ch` and `comm-java.ch`. If `ctangle` and `cweave` are compiled with these change files (as, for example, by modifying the `cweb` `Makefile` by changing the line `TCHANGES=` to `TCHANGES=ctang-java.ch`, and similarly for `WCHANGES` and `CCHANGES`). then the `+j` switch[2] can be used to output JAVA:

```
% ctangle +j DVI.w
```

produces the file `DVI.java`, which can then be compiled in the usual way

```
% javac DVI.java
```

On the other hand, the documentation is produced by

```
% cweave +j DVI.w
```

creating the LATEX file `DVI.tex`, which can then be processed in the usual way

```
% latex DVI
% xdvi DVI
% dvips DVI
```

Ctangle. In passing from `web` to `cweb`, Knuth and Levy dispensed with the macro feature `@d`, on the grounds that its functionality was more than adequately provided by C's `#define`.

However, JAVA in turn has dispensed with `#define`, so it seemed useful to transplant back this lost feature from `tangle` to `ctangle`. Fortunately, this turned out to be relatively simple since the amputation had been crude and the stumps remained. This allows us, for example, to say in `DVI.w` (and elsewhere)

```
@d DviUnit == int
```

and then

```
void moveRight( DviUnit dh );
```

This clarifies the code and also makes it simpler to change the type of `DviUnit` if that should prove desirable.

Cweave. The changes to `cweave`, although more trivial, proved surprisingly tricky. The problem is that `cweave` (like `weave`) is based on a table of "productions" — a kind of pseudo-syntax which allows scraps of code to be "reduced". It turned out that JAVA required some five new production rules to add to the 100 or so rules for C . . .

Web2java. `Web2java` — like `web2c` — is a post-processor to `tangle`. To create `foo.java` from `foo.web` and `foo.ch` one first runs `tangle`:

```
tangle foo.web foo.ch
```

[2] The use of `+` rather than `-` as a prefix for switches is a feature of `cweb`.

This creates the PASCAL (or pseudo-PASCAL) file `tangle.p` (or `tangle.pas` on some systems).[3] Class files are machine-independent — provided "native methods" are eschewed, and care taken to avoid such OS-specific idioms as '`\n`' for end-of-line, rather than JAVA's '`line.separator`' — so `tangle.class` from the JAVATEX distribution should run on any system. Note that this file, like all JAVATEX programs, is defined to be in the `javaTeX` package, and so must be placed in a subdirectory called `javaTeX` relative to the `CLASSPATH`. Note too that JAVA refers to this class as `javaTeX.tangle`, rather than `javaTeX/tangle`, as one might expect.

This file is then passed through `web2java` to create `foo.java`:

$$\texttt{foo.web} + \texttt{foo.ch} \xrightarrow{\texttt{tangle}} \texttt{foo.p} \xrightarrow{\texttt{web2java}} \texttt{foo.java}.$$

Actually, this is a slight oversimplification. The file `common.defines` is prepended to `foo.p` *before* passing through `web2java`:

$$\texttt{common.defines} + \texttt{foo.p} \xrightarrow{\texttt{web2java}} \texttt{foo.java}.$$

All this is completely analagous to `web2c`, except that we are able to dispense with the post-processor `fixwrites`, for JAVA I/O contains nothing as exotic as C's `printf`.

The filter `web2java` is created by the programs `flex` and `bison` (or `lex` and `yacc`) from the files `web2java.l` and `web2java.y`. This is completely analagous to `web2c`. The `lex/flex` file `web2java.l` is the same as `web2c.l`, with the addition of a small number of new tokens: `new`, `try`, `catch`, `throw`, `throws`, etc. The syntax description in `web2java.y` has rather more changes, compared with `web2c.y`.

On the plus side, since JAVA has no pointers all the pointer-related material has been deleted. There is no attempt to determine if a function argument is "formal var" or not; and no need, therefore, to re-name functions with such arguments.

On the other hand, the introduction of class and object tokens necessarily adds to the number of rules in `web2java.y`. Thus variables and functions can be preceded by a `CLASSIFIER`, consisting of a (possibly empty) sequence of `class_id_tok`s and `object_id_tok`s followed by periods ('.'s). For those familiar with `web2c`, a short excerpt from `web2java.y` should give the idea:

```
CLASSIFIER:
  /* empty */
| CLASSIFIER class_id_tok '.'
  {
    my_output(last_id);
    my_output(".");
  }
| CLASSIFIER SIMPLE_OBJECT '.'
  {
    my_output(".");
  }
;

SIMPLE_OBJECT:
  object_id_tok
    {
      my_output (last_id);
    }
  VAR_DESIG_LIST
| object_id_tok
  {
    my_output (last_id);
  }
;
```

But for the most part translating `WEB` to JAVA is, if anything, simpler than `WEB` to C. One apparent difficulty is the lack of a pre-processor in JAVA, since `web2c` leaves a good deal of work to `cpp`. This means that more must be done in the change file, which is probably A Good Thing. The three main issues which arise are:

- the absence of `goto`s in JAVA
- the lack of `typedef`s in JAVA
- input/output

These are discussed briefly in the following three subsections.

Removing `goto`s. Java has no `goto`; in compensation, it allows `break` and `continue` statements to carry a *label*, as in `break lab21` or `continue lab3`, for example. The corresponding labels must appear at the beginning of the loop in question. (A `break` label can also be attached to a `switch` statement, but we make no use of that.) If a `break` or `continue` statement has no label, it is understood to refer to the smallest loop (or switch) enclosing the statement. Thus, labelling is only required in the case of nested loops or switches.

Fortunately, Knuth has followed a strict protocol in the classic `WEB` files. Raw `goto`s (as in `goto 40`) very rarely appear. In almost all cases a label is used, as in `goto found`, where `found` has earlier been defined as

```
@d found=40
```

In effect, the `goto`s are divided into a small number of cases, according to their function. By far the most frequent of these cases are: `goto break` to break out

[3] If you like driving in the slow lane, you could run the JAVA `tangle` instead: `java javaTeX.tangle foo.web foo.ch`.

of a loop, `goto continue` to continue around a loop, and `return` to return from a routine.

This protocol allows most `goto`s to be processed automatically. Thus `goto break` is translated as `break` and `goto continue` as `continue`, while `return` is translated as `return` (with the appropriate value in the case of a function).

However, in perhaps a third of the `goto`s, labels must be inserted by hand, for example, as a `break` out of an outer loop. Note that in such a case the label in the `WEB` file is almost certainly in the wrong place, for, by Knuth's convention, `break` means "break to the end of the loop" while JAVA requires the label to appear at the start of the loop. `web2java` takes advantage of this by deleting the label from a `break` or `continue` statement unless that label has already appeared (in the current routine) before the statement.

Of course a `goto` may not go to the beginning or end of a loop; in that case a new "artificial" loop must be inserted, with a `break` at the end to ensure that it is only traversed once.

All this is rather messy and could probably be automated to a much greater extent. At least some checks have been introduced in `web2java.y`, to verify as far as possible that the new code has the same effect as the old.

Type definitions. There are no `typedef`s in Java. In theory one could replace `typedef`s by class definitions but that would add considerable complication to the code. Instead, we simply change them to substitutions (as though in C changing `typedef`s to `#define`'s). So, for example, we make the change

```
@x
@<Types...@>=
@!ASCII_code=0..255;
@y
@d ASCII_code==0..255
@z
```

Later `web2java` will replace this range `0..255` by an appropriate type (currently `int`). This entails some changes in `web2java.y`, to allow ranges for procedure and function parameters such as:

```
procedure p(x:0..255);
```

Presently all ranges are replaced by `int`, since Java is rather strict about type conversion, and requires casting where C does not.

Input/Output. JAVA I/O is much closer to PASCAL syntax than is C. Thus

```
write_ln(term_out, 'value is ', v);
```

in PASCAL becomes

```
System.out.println("value is " + v);
```

in JAVA. This allows us to incorporate I/O into `web2java.y`, dispensing with the `fixwrites` postprocessor required by `web2c`.

The only unusual feature of JAVA I/O is that most I/O statements must be contained in a `try` statement, which in turn must be followed by a `catch` statement to catch any I/O 'errors'. However, this is perfectly straightforward, as may be illustrated by an I/O function from `DviReader.ch`:

```
function signed_pair:integer;
  {returns the next two bytes, signed}
var a,@!b:eight_bits;
begin a:=0; b:=0;
try begin a:=dvi_file.readByte;
  b:=dvi_file.readUnsignedByte; end;
catch (ex: IOException) begin
  EOF_dvi_file:=true; end;
if EOF_dvi_file then signed_pair:=0
else begin cur_loc:=cur_loc+2;
  signed_pair:=a*256+b; end;
end;
```

Conclusion

Hopefully, this all-too-brief tour has given some taste of the JAVATEX project. All comments, contributions and suggestions will be gratefully received.

The project (and all its parts) is freely available (Murphy, 1999). For simplicity it is published subject to the GNU GPL Licence, Essentially this allows the work to be freely copied and used, provided the original files `DVI.w`, etc, are made available. Changes should preferably be made through change files, e.g., `DVI.ch`.

References

Etymon Systems. "Java software for parsing, manipulating, and creating Adobe PDF file". `http://www.etymon.com/pj/`, 1999.

Knappen, Börg. "NTS-FAQ". `CTAN/info/NTS-FAQ`, 1995. (In these references, CTAN denotes any of the CTAN sites, eg `ftp://ftp.tex.ac.uk/pub/tex` or `ftp://ftp.dante.de/pub/tex`).

Knuth, Donald E. *Literate Programming*. CSLI, Stanford, 1992.

Murphy, Timothy. "The JAVATEX project". `http://www.maths.tcd.ie/~tim/javaTeX`, 1999. Also available from `CTAN/systems/java/javatex`.

Sun Microsystems. "Java HotSpot Performance Engine". `http://java.sun.com/products/hotspot/`, 1999.

Barbara N. Beeton: TUG Board Member for 20 Years

Christina Thiele, Proceedings Editor

Abstract

Barbara Beeton has been a board member for twenty years, since 1989/90, when she was listed as 'Wizard of Format Modules' on the TUG Steering Committee — the info's on cover 2 of the very first *TUGboat* issue, 1(1).

As the only board member to have 'been there' since the beginning, Barbara has seen TUG presidents come and go — seven in all. And so the idea came to me that surely some of us would have something to say about attending board meetings with Barbara every summer (and one winter — Cincinnatti 1982) for the past twenty years. Barbara, these pages, for a change, are about you!

Pierre MacKay (1983–1985)

Is it possible that there was a time before I could count Barbara as a friend? The calendar tells me that there has to have been such a time, but the calendar is oddly unconvincing. When I first arrived at a meeting of the TEX Users Group (only a couple of sessions after Barbara had led the initiative to create it) I was surely the most naïve and inexperienced of all the attendees who were to become site-coordinators, but I seem to remember that when I talked to Barbara I came away with the impression that I knew what I was talking about. There not many people with the talent for instant and lasting friendship that Barbara offers to those who make the effort to recognize it. My earliest specific memory is a discussion, by mail, of the virtues of an old Corona typewriter with misaligned punctuation, on which I submitted my first, rather irrelevant, contribution to *TUGboat*. In that correspondence it already seemed as if I had always been a member of TUG, and the feeling has remained, although the calendar again tells me it cannot quite be the case.

Past Presidents

Richard Palais	(1980–1983)
Pierre MacKay	(1983–1985)
Bart Childs	(1985–1990)
Nelson Beebe	(1990–1991)
Malcolm Clark	(1991–1992)
Christina Thiele	(1992–1995)
Michel Goossens	(1995–1997)
Mimi Jett	(1997–2003)

I can't imagine what my term of office as president of TUG would have been without Barbara. As everyone knows who has filled the office since, it is partly a sinecure as long as Barbara is there. And after the business is over there is the time spent talking of everything that friends can talk of. That conversation never ends; it only adjourns, ready to be picked up again at the next meeting.

Nelson Beebe (1990–91)

I've been meeting Barbara Beeton for almost 20 years now at TUG gatherings, and I never cease to marvel at her dedication to TEX, to TUG and its Board, to *TUGboat*, and to all things typographic. She has my deepest thanks for all her work; it has always been a great pleasure to work with her.

Barbara has been with *TUGboat* right from the beginning, succeeding Robert Welland as Editor-in-Chief with Vol. 4, No. 2 (Sept. 1983). In March 1999, *TUGboat* Vol. 19, No. 1, reached a milestone of 2,000 published articles. More than 1,900 of them have appeared since she took the helm, and the inky waters have been typographically rough and challenging: I don't know of any other journal which has published articles with so many different fonts, and from so many output devices. There are 100 *TUGboat* articles that bear her name, 96 of them with her as the sole author. There are also 674 short articles credited to Anonymous, the bulk of which I believe are her creations as well. *TUGboat* is always interesting, and I look forward to every issue.

Barbara has made, and continues to make, important contributions beyond the TUG community, with her long involvement as a representative of the American Mathematical Society in the international standardization of mathematical character sets.

She is also the sole channel for reports to Don Knuth on TEX and METAFONT bugs, problems, and suggestions, thereby helping to shield him from distractions that would further delay *The Art of Computer Programming* series that, we should remember, was the reason that TEX and METAFONT were written in the first place.

Barbara has probably attended more TUG and TEX conferences than anyone, and as a result, is probably the world's expert on what new things people around the world are doing with TEX and METAFONT.

Don't ever retire, Barbara! We need you.

Past Meetings

1980	Stanford, Calif.
1981	San Francisco, Calif.; Stanford
1982	Cincinnati, Ohio; Stanford
1983	Stanford
1984	Stanford
1985	Stanford
1986	Medford, Mass.
1987	Seattle, Wash.
1988	Montreal, Canada
1989	Stanford
1990	College Station, Tex.; Cork, Ireland
1991	Dedham, Mass.
1992	Portland, Ore.
1993	Birmingham, UK
1994	Santa Barbara, Calif.
1995	St. Petersburg, Fla.
1996	Dubna, Russia
1997	San Francisco
1998	New York, NY; Toruń, Poland
1999	Vancouver, Canada

Christina Thiele (1992–95)

I don't remember when I first met Barbara. In fact, my first meeting in Seattle (1987) was somewhat of a blur once I gave the opening talk (some 10 minutes faster than I'd clocked it). But I must have met her.

The following year, in Montreal, I joined the TUG board (Bart was president), and over time I learned a great many things I'd never have learned anywhere else. For me, moving up from 'just a board member' to member of various committees, and then on to the executive — Barbara's been the best constant factor I could ever have imagined.

She remembers things. She knows the right things to do. As much as she knows TEX, she's knows procedure! And while I still can't seem to take in much of what she tells me about TEX, I most certainly have taken in an awful lot about procedure: how to work within procedures, how to be very careful when adopting procedures, how to suggest when procedure is useful and when it's just a constraint.

For me, Barbara represents the collective memory of TUG; she is our most valuable resource and she is one of the best friends I have made during my own adventures in the TEX community.

Michel Goossens (1995–97)

It was in July 1988 at the Third EuroTEX Conference in Exeter that I first met Barbara Beeton, that "funny American woman with the wide hat." When somewhat later I also met Joachim Schrod with his famous cowboy-like hat, I really started thinking that all those TEX people were weird indeed . . . [1]

Of course, the name of bb was not completely unknown, since I had seen it on the front cover — and in various other places — in *TUGboat*. So there I was, speaking to the living legend, the person who had a direct line to Knuth himself. And, although I myself and a lot of the other participants were only novices in TEX, Barbara took all her time to gently explain, with the necessary detail and with eternal patience, this or that trivial or not-so-trivial point about TEX. Quite an experience for my first TEX conference and without a doubt this helped convince me to get to know more TEX and friends.

Later, when I got more involved with the practical day-to-day support of TEX and became a board member of both GUTenberg and TUG, I had the occasion to meet Barbara more often and got the opportunity to appreciate other aspects of her multi-faceted personality. Barbara has a special sense for listening to what people have to say, and for trying to build a consensus. She draws on her many years of experience dealing with people in the TEX world, where she is well known and respected, but, more importantly, where she knows almost everybody personally.

As a Continental European, and probably the only non-native English-speaking president of TUG, I came to appreciate the importance that Barbara attached to contacts with TEX users all over the globe. Thus, she always did her best to attend TEX conferences in Europe or in North America, supported the creation of local user groups and promoted the exchange of information, publications, etc. I consider Barbara to be a genuine example

[1] I found out later that Barbara and Joachim shared another passion: gastronomical outtings and visiting famous wine cellars.

what an international collaborative effort could and should be.

To me, Barbara is the ideal safekeeper of the history of TEX and TUG, one of the few who were present "from the very beginning" — and are still around to tell us the story. Thus, she is ideally placed to remain the Voice of TUG and TEX well into the next century, and I look forward to her wise words in the Editor's note: of *TUGboat* at least until the year 2010.

(Photo by Warren Leach, Blue Sky Research)

Mimi Jett
(1997–2003)

There are some people in this world who are so unique, once you meet them you never forget. They have a particular style and demeanor that separates them from the crowd, subtle but brilliant. Barbara Beeton is just so unique. My guess is that most people who have met her will agree — something about that meeting is memorable, special.

I first saw Barbara at the 10th annual meeting in 1989 at Stanford; however, we did not meet until the following year in TEXas, when I started becoming aware that this was not only a collection of some interesting characters, TUG was clearly an important organization. If people like Barbara, Bart Childs, and Pierre MacKay were willing to donate so much time and intellect to this, it must have extreme value. Within another year, I was involved with the conference program committee and soon the board.

The importance of Barbara's contributions during my years with board cannot be quantified. She is the voice of reason, sometimes our conscience, but always the expression of objective, non-judgemental truth. During the most heated discussions or the boring details of by-law semantics, Barbara is the one person who can consistently separate the wheat from the chaff and give us a sense of having accomplished something. There've been times of frustration when she has pulled me through with her patient friendship.

Knowing how many people share my appreciation for her friendship, it is a wonder she has time for any work at the AMS, with such a heavy schedule for support for all of us.

In Vancouver this summer, I was the fortunate driver of a busload of hungry TUGies; I looked in the rearview mirror and realized I had some of my favorite characters on board: Barbara, Christina, Michael Doob, Pierre, Craig Platt, Don DeLand. It felt like all these years had led us to that moment. Vancouver was our 20th annual meeting, and Barbara was recognized for her decades of service with a bottle of Russian Vodka, imported by Irina Makhovaya. There is no way to thank you, Barbara, for 20 years of service on the board, except to say "Thanks, and would you mind another 20?"

POSTER EXHIBITION:

Text of *The Apocalypse* as Graphics by Prof. Alban Grimm

Christina Thiele

The Alban Grimm exhibit, 'Text of the Apocalypse as Graphics,' on display during the TUG'99 meeting, shows what happens when a typographer-cum-graphic artist is introduced to computer programs and finds that inputting code can lead to outputting incredible visual results.

At the urging of his son, Gerhard, Professor Alban Grimm first encountered TEX and METAFONT via AmigaTEX, so that Gerhard's thesis in electrical engineering might be typeset. Browsing through *The TEXbook* and *The METAFONTbook*, the professor recognized the vast possibilities that TEX offered for his own field of work — and he was motivated enough that he even attempted to overcome the language barrier while studying those books. Several months later, he meet Frank Mittelbach for the first time (summer of 1990).

One of Prof. Grimm's aims was to study the nature of computer-generated type. That is, a hand-made type has the look and feel of hand-made type; if one used a chisel, characteristics specific to chisels would be apparent. So — can one extend this idea and identify features of truly computer-generated type (in contrast to type generated with the computer but emulating other methods)? The 'VN' set of font variations which resulted were inspired by a search for a font to set the biblical text, *The Apocalypse* by St. John.[1]

It was Frank who suggested that an exhibit of this melding of computer code and graphic genius be held at TUG'99. We are therefore deeply indebted to him for provoking Prof. Grimm with METAFONT, and for making the arrangements to bring over from Germany the enormous folder containing these posters (a weight of over 10kg!). Fortunately, Wendy McKay and Ross Moore (also responsible for the TUG'99 web version of this material) were able to pick up both Frank and the folder from the airport.

[1] A revelation made to St. John and recorded by him in the last book of the New Testament, called also the *Book of the Revelation of St. John the Divine*.

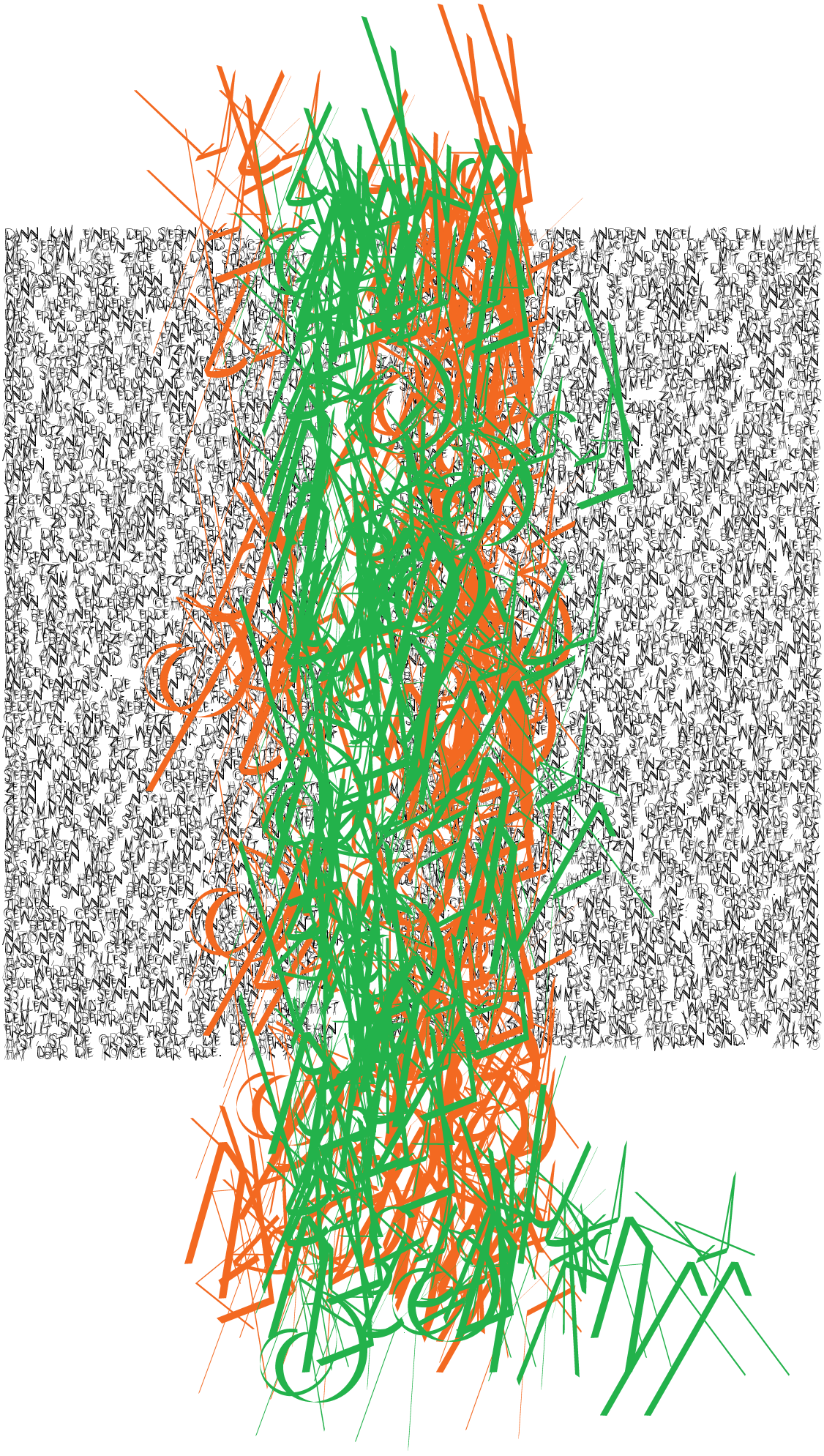
The exhibition comprised 7 sets of panels, each set being the full text (in German) of the *Apocalypse*. Some sets had 22 graphic pages while others only 11 (where two chapters were set on one page). In all, 132 panels were displayed, with Frank putting up a different set each morning and afternoon, with assistance from Anita Hoover, program co-chair and technical editor of the proceedings. In addition, several texts written by Prof. Grimm and translated by his son provide some background to the whole project.

To describe the images in this exhibit is nigh impossible. Almost. Imagine a large off-white poster-sized panel, with two main areas of 'stuff' rendered in varying combinations of black, red, blue, yellow, ... The upper area shows material that is text (it just doesn't look like text), while the lower area (in the early series) is clearly an all-caps text (the VN font). The graphics on top of each page were generated from the first and last sentences of the current chapter, which appears in the lower area (either on one page or two). The difference in presentation is the result of selecting appropriate METAFONT modifications.

The poster included in these proceedings has black text in the background (the 'square' configuration), and then red overlaid with bright green (lighter gray) in the foreground. Of course, these black and white reproductions cannot fully do justice to the work; I invite you, therefore, to follow the links from `www.tug.org` for a better view.

Obviously a colour monitor is a good idea (!). Since these are image files, downloading takes a while; in addition to `.jpg` format, the `.pdf` versions of the multi-colored images provide an additional show, as the image 'develops' on-screen.

Prof. Grimm generously allowed all his prints for this exhibit to be given away to the participants at TUG'99. We would like to express our thanks — and our amazement — to Prof. Grimm, for having shown us that paper and ink can go far beyond 'boxes and glue'.



Text of *The Apocalypse* as Graphics

Prof. Alban Grimm
Joh. Gutenberg-Universität Mainz
Fachbereich 24
Am Taubertsberg 6
D–55099 Mainz
Germany

When undertaking graphical experiments on the text of John's Apocalypse over and over, I am always fascinated with the interaction of shape and content which this text stimulates. To my eyes, reading about such horrible events via a font of neutral style, one which would be appropriate for any purpose at all, is disturbing. The conflict between shape and content irritates me. By attempting to compensate for the lack of specific suitability solely by employing appropriate arrangements, I learned that the existing variations of our roman typefaces are dedicated to an aesthetic that is purely self-related.

My basic interest in creating an interaction of text and contents of John's Apocalypse in manifold ways has nothing to do with a fashionable end-of-times mood. The more I give in to previously unknown possibilities of varying our typefaces, the more I find confirmation of the fact that script as shape extends beyond itself. This is very suitable for the text of John's Apocalypse.

If images — even excellent ones — remain hidden behind the telling of the Apocalypse, one has to try to render the text itself as an image. This relieves the artist from the possibility, as well as the constraint, to provide illustrations. Associations which resemble illustrations are illusionary. This is not about illustrations. The text itself moves into the pictorial domain — those transcriptions can, however, never be interpreted as illustrations. The script merely visualizes itself, being exposed to conditions that are no longer dedicated to legibility.

Due to the nature of roman capitals, they lend themselves easily to such transformations. A special appeal results from the enhancements with digital graphical elements, and astonishing results beyond legibility can be achieved. Structures consisting of very different computer-specific strokes resemble hand-drawings but, although they are static, they exhibit a vivid appearance of a completely different sort.

Most of the attempts are concerned with the contrast of script as text versus script as a language-free play of shapes. Beyond the domain of language, everything can be different. The process of reading is no longer tied to running along the lines with their sequence of words. Relieved of the constraints of reading, the eye can move here and there, can follow the scattering and clustering of lines back and forth, up and down. The irreversibility of the unidirectional nature of reading, a function of correct linguistic interpretation, is obsolete. Furthermore, reception is uncoupled from the semantics of language. The transcriptions can be concentrations, where reading reverts to the Latin *legere*.

The liberated characters form something new, still requiring the text as a base, and thus challenge the recipient's thoughts to respond to this unusual play of lines, according to his readiness and ability. Even rejection can be explained: those who regard writing only as a cultural technique, something learned in school, cannot be expected to be very open-minded towards these transcriptions. Such transcriptions are monstrous in terms of linguistic functionality, because they make use of text in non-standard ways. Being unreadable and thus exclusively graphical, the text is only related to itself and is itself the subject of the various visualizations.

Thus, as an observing and thinking being the reader is referred back to himself, his willingness to reflect encouraged.

META FONTS BY METAFONT

The variable capital font VN is only available as a program capable of scaling various types. The range of variance is unimaginably large and cannot be exhaustively demonstrated by examples. The interaction of 21 parameters controlling the type creation of this font is being reduced here to the contrast of writing as text versus writing as graphics.

This basic example shows the isolated basic shape with a constant stroke width.

Example A uses a narrow broad-nip with unusual, varying angles and small deviations from the basic shape.

Example B uses a broad-nip with varying width and emphasizes the deviations from the basic shape. The character heights also differ more distinctly.

Example C uses several traces, each one repeating the character shapes differently. The broad-nip is changing, and the heights are strongly differentiated.

Example D employs, in addition the usual character trace, another trace which is only partly aligned with the corresponding shape. This “sub-trace” creates here a multitude of light strokes, especially deviant at roundings.

Example E does not use broad-nip and allows the character shapes more deviance from the basic shape than the sub-trace, consisting of bundled dots. As in example D, the alternative sub-trace is a graphical enhancement.

WORKSHOP:

LaTeX to XML/MathML

Eitan Gurari
Ohio State University
gurari@cis.ohio-state.edu

Sebastian Rahtz
Oxford University, UK
s.rahtz@oucs.ox.ac.uk

The objective of this workshop was to show what it takes to translate LaTeX sources into XML in general, and LaTeX mathematics into MathML in particular. In addition, it aimed at reviewing backward translations from XML and MathML into TeX and LaTeX. Many of the details can be found in *The LaTeX Web Companion*.

We started by demonstrating the viability of such translations. We provided pointers to 38 source files in the public domain, including AMS preprints, and their corresponding outcome in a XML dialect consisting of the union of XHTML and MathML. Then we looked at an example of a backward transformation, which had been used to create PDF output. As a side product, we noted that translations to MathML may be used for debugging LaTeX formulae. The translations relied on `TeX4ht` for the forward direction, and PassiveTeX for the backward direction.

The second part of the workshop reviewed XML as an evolution from HTML, demonstrated the use of Cascading Style Sheets (CSS) for specifying the look of XML code, and illustrated the application of XSL for defining transformations on XML documents.

In the third part we took a look at how the translation of LaTeX documents can be managed, for HTML, XML, and MathML output. Beyond the use of built-in modes, we dealt with user configuration of the output, and the constraints imposed by the TeX engine and LaTeX style files. Finally, approaches and tools for backward translations were reviewed.

We concluded by considering the relationships between LaTeX, TeX, and XML, and calling for more coordinated work within the LaTeX community.

References

[1] Slides of workshop:
`www.cis.ohio-state.edu/~gurari/tug99/`

[2] *The LaTeX Web Companion*, Michel Goosens and Sebastian Rahtz, with Eitan M. Gurari, Ross Moore, and Robert S. Sutor. Addison Wesley Longman, 1999.

[3] `TeX4ht`:
`www.tug.org/applications/tex4ht/mn.html`

[4] PassiveTeX:
`users.ox.ac.uk/~rahtz/passivetex/`

WORKSHOP:

How to Create Quality Interactive PDF Documents for the WWW Using LaTeX

D.P. Story
Department of Mathematics and Computer Science
The University of Akron, Akron, OH 44325
dpstory@uakron.edu
http://www.math.uakron.edu/~dpstory/

This workshop covered many concepts that go into the creation of quality *interactive* PDF documents for the WWW. Topics included:

- setting up the Acrobat/TeX System (a Win95 based presentation)
- page layout for a PDF document designed to be read over the Web
- adding a dash of color to text and background
- using `hyperref`, a package by Sebastian Rahtz, and its "∞-many" options — only finitely many were discussed.
- macros to create (1) problem exercises with hyperlinked solutions; (2) multiple-choice questions with instant feedback; and (3) multiple-choice graded quizzes using JavaScript.

Extensive supplementary material in the form of paper hand-outs and software (which included electronic technical articles and macro packages) were distributed. These materials are available from the AcroTeX web site:

`www.math.uakron.edu/~dpstory/acrotex.html`

From TeX to PDF

A discussion of various methods of creating quality PDF documents from a TeX or LaTeX source file. These comments are contained in my paper "AcroTeX: Acrobat and TeX Team Up" (elsewhere in these proceedings); in particular, see Figure 1.

Win32 TeX systems

The workshop was oriented towards Win95/NT operating systems, although most comments were platform-independent. A brief mention of TeX implementations designed for the Win95/NT operating system and capable of producing quality PostScript output using Type 1 fonts: the Y&Y system, the MikTeX system by Christian Schenk and the fpTeX system by Fabrice Popineau (see elsewhere in these proceedings for Popineau's paper).

Quality PDF documents for the WWW

The content of this segment of the workshop is contained largely in the electronic article, "Using LaTeX to Create Quality PDF Documents for the WWW", available at the AcroTeX web site; a printed version of this article was also distributed.

The `web/exerquiz` packages

The `web` package redesigns the page layout to a more Web-friendly style, suitable for screen viewing.

The `hyperref` package provides a high degree of interactively through hypertext links and form elements. The `exerquiz` package uses `hyperref` to define several environments for creating on-line exercises and quizzes.

The capabilities and features of these two new packages were demonstrated; manual of usage and the packages themselves were distributed.

Since the time of the workshop, several new features of the `exerquiz` package have been added; most importantly, quizzes defined by the `quiz` environment can now be graded *and* corrected using JavaScript.

A do-it-yourself tutorial

A do-it-yourself tutorial on `hyperref` and the `web` and `exerquiz` packages was made available on a few discs distributed at the workshop. The material is now available at the AcroTeX website, see the link "TUG99 Handout Material".

Concluding remarks

I was pretty well satisfied with the course of the workshop; I was able to cover all the advertised topics and give several demonstrations on the computer. I was very happy to see that the workshop was well attended and well received. Judging from the numerous questions posed after the workshop, there is quite an interest in this topic. Thanks to all the attendees for your kind response to my workshop (and to my talk).

WORKSHOP:

Writing Class Files: First Steps

Michael Doob
University of Manitoba,
Winnipeg, Canada
mdoob@cc.UManitoba.CA

The workshop covered (just) enough background to allow participants to write their own class files, using the standard file `classes.dtx` as a model. Topics included:

- class files and the `docstrip` concept
- the different file extensions — an alphabet of woe?
- the `classes.dtx` file
- adapting the standard to make your own class file

A handout was provided, showing how a class file was created for the Canadian Mathematical Society's publications, the *Canadian Mathematical Bulletin* and the *Canadian Journal of Mathematics*.

These included:

1. the file `cms.ins`
2. output from running `cms.ins`
3. the driver part of `cms.dtx`
4. the file `cms.drv`
5. the file `classes.ins`
6. a macro from `cms.dtx`
7. the same macro in `cms.cls`
8. the same macro in the documentation

WORKSHOP:

Converting a LaTeX 2.09 Style to a LaTeX 2ε Class

Anita Hoover
University of Delaware
anita@udel.edu

The workshop was well attended. There were over 55 people in attendance; a handout with an outline of the process was also provided. The main focus of the workshop was to provide enough information to begin converting an old LaTeX 2.09 style file into a LaTeX 2ε class file. The objectives included:

- pointers to available documentation
- converting existing 2.09 style files to a class or package
- conversion steps

Documentation

There is a lot of documentation available with the distribution of LaTeX 2ε (see `texmf/doc/latex/base`); those most helpful for converting style files to classes are:

- "LaTeX 2ε for class and package writers" (file: `clsguide.tex`)
- "LaTeX 2ε for authors" (file: `usrguide.tex`)

Additional books for specific issues include *The LaTeX Companion*, by Goossens et al., and Lamport's *LaTeX: A Document Preparation System*

Converting a style file to a class or package

General rule of thumb: if the commands can be used with any document class, then put them into a package; if not, then put them into a class file.

Conversion steps

- Does it run in compatibility mode?
- Does it depend on another style?
- Structure setup.
- Make it robust.

The workshop applied these steps to the University of Delaware thesis style file (`udthesis.sty` and `udthe12.sty`). Attendees were able to see the process of transforming the style file into a new class file, `udthesis.cls`.

Conclusion

The example of transforming `udthesis.sty` into `udthesis.cls` clearly showed how to convert a style file into a class file, where the original style was based on one of the standard style files such as `book`, `article`, or `report`.

The point was also raised that style files which had been built from a combination of many style files would be much more difficult to convert easily to a class file and most likely this would require starting from scratch.

Resources

There is a Powerpoint slide presentation

`LaTeXstyle2class.ppt`

available via anonymous ftp for download from

`zebra.us.udel.edu`

in

`pub/tex/TUG99/workshops/hoover`

Also in this directory are the University of Delaware thesis files, `udthesis.sty` and `udthesis.cls`.

Access from outside the University of Delaware is limited to the hours of 6:00pm to 8:00am (Eastern Standard Time) Monday through Friday, and all day on the weekends.

PANEL DISCUSSION:

TEX and Math on the Web

Stephen A. Fulling, moderator
Mathematics Dept.
Texas A&M University
College Station, Texas
77843-3368 USA
fulling@math.tamu.edu

Panelists:

- David Carlisle, LATEX3 Project (UK)
- Michael Downes, American Mathematical Society
- Andre Kuzniarek, Wolfram Research
- Jeffrey McArthur, Atlis Publishing Services
- Ross Moore, Macquarie University (Australia)

Moderator's summary of views

The moderator started the discussion by asking how soon his non-negotiable demand for math symbols on the Web would be met.[1]

Various panel members reported that *partial* solutions are provided by PDF, Scientific Word, Techexplorer, Publicon, and MathType, and that the Netscape-affiliated Mozilla Organization will soon provide Windows rendering of MathML (albeit typographically poor at the moment).

McArthur pointed out that searching and indexing of the contents of PDF documents is not currently possible. This set off a lengthy colloquy among various members of the audience and panel on whether indexing of mathematical expressions makes practical sense in the first place.

McArthur said that TEX should be fixed to emit XML, and its cousins. From the audience, Sebastian Rahtz stated that Ω already does this. Carlisle observed that sub-expressions are hard to handle.

The key need, said audience member Art Ogawa, is MathML rendering in the browsers. Carlisle replied that math symbols will soon be incorporated into UNICODE (as a tiny perturbation on its linguistic riches), and it will then be easy to map them into existing font sets. Kuzniarek pointed out that the *Mathematica* fonts are freely available. Don DeLand raised the issue of server vs. client support for fonts.

[1] The panel discussion was based on the 13-point "Dreams and Difficulties" handout provided by the moderator. -Ed.

McArthur suggested that TEX can be treated as a language, like Chinese, for which input editors exist. The editor could convert to MathML, and also convert backwards to something editable. Kuzniarek said that the translation might be trickier in this case, but Peter Flynn replied that the Euromath Grif [object-oriented editor recently adopted by the Euromath consortium] already performs such conversions adequately.

Timothy Murphy and Michael Doob predicted that most mathematicians will stick with TEX, no matter what; mathematics is a separate world, which TEX serves very well. These comments provoked a spate of "on-the-other-hand" remarks:

- Carlisle: TEX users need to get onto the Web somehow.
- Patrick Ion: Engineers at Boeing (for example) use math too, and they need to read and write it.
- Fulling: We can't reach our students if they encounter mathematics only in an environment that is alien to them.
- McArthur observed that TEX has surprising difficulty in dealing with elementary-school math.

Ogawa summarized the task before: Both rendering and document creation are crucial needs, and both will be hard sells as the small TEX community struggles to integrate itself into the XML/MathML world.

PANEL DISCUSSION:

TEX in Publishing

Siep Kroonenberg
Kluwer, Dordrecht
siepo@cybercomm.nl

Panelists:

- Kaveh Bazargan (moderator), Focal Image Ltd. (UK)
- Fred Bartlett, Springer NY
- Jean-luc Doumont, JL Consulting (Belgium)
- Nadia Molozian, Harcourt Intl. (UK)
- Sebastian Rahtz, Oxford University (UK)

Summary of views

Points raised during the day's panel discussion:[1]

- Nadia Molozian from Harcourt Publishers noted a strong increase in the use of LATEX in production at her company. An advantage of LATEX is that copy editing involves less work.
- Generally, LATEX submissions by authors also appear to be up, although this is not true everywhere.
- Production of conference proceedings is a messy business; often, quick-and-dirty measures such as photographic resizing must provide a semblance of consistency.
- The publisher has little chance of influencing the coding style of monographies. Often, the author has been working on his book for years before a publisher gets his hands on it.
- An interesting speculation by Frederick Bartlett on why authors like to use bad LATEX coding: writing is hard work; authors cast about for distraction and find it in fiddling with appearances.
- The same speaker encouraged the audience to complain to publishers about bad-looking books; this would give publishers an incentive to let their TEX specialists do something about it.

[1] This summary was first published in *MAPS*, the communications of the Dutch User Group NTG, Number 23 (1999), pp. 10–11, and appears by kind permission of the NTG editors and the author. This text is part of an overall summary of the TUG99 meeting, which appears in the same issue (pp. 8–12).

PANEL DISCUSSION:

Future of LaTeX

Arthur Ogawa, moderator
TeX Consultants, California
ogawa@teleport.com

Panelists:

- Donald Arseneau, TRIUMF (Canada)
- Fred Bartlett, Springer NY
- David Carlisle, LaTeX3 Project (UK)
- Michael Downes, American Mathematical Society
- Steven Grathwohl, Duke University
- Andre Kuzniarek, Wolfram Research
- Frank Mittelbach, LaTeX3 Project (Germany)
- Jeffrey McArthur, Atlis Publishing Services
- Ross Moore, Macquarie University (Australia)
- Chris Rowley, Open University (London)

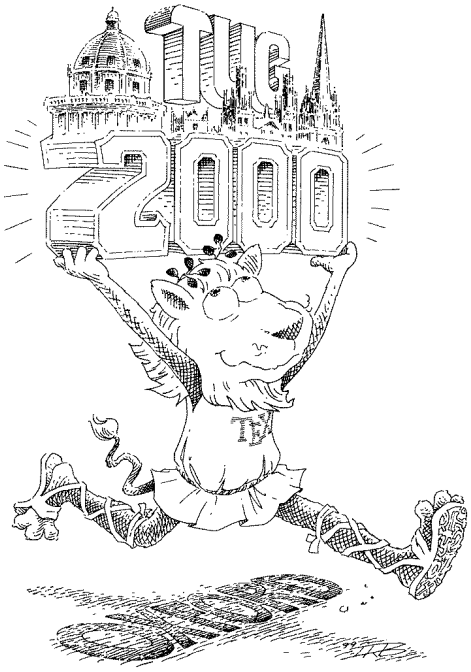
TUG 2000

Wadham College, Oxford, UK
August 13th–16th, 2000

The 21st Annual Conference of the TEX Users Group will take place at Wadham College, Oxford, between Sunday 13th August and Wednesday 16th August 2000. Tutorials will be given on the 17th and 18th August.

The Location

Oxford is a small, pleasant city with an internationally famous university. The city is full of ancient buildings, beautiful gardens, libraries and bookshops. The conference will be held in Wadham College, a traditional college (founded 1613) in the centre of the city. Oxford is easily reached from London, and is a good starting point for visiting much of southern England.



The Conference

The conference will feature talks on all aspects of TEX and its relationship to both traditional and electronic document preparation and processing.
The Annual General Meeting of the TEX Users' Group will be held during the period of the conference.
We expect the cost to a typical delegate to be about £300, including accommodation and meals; cheaper accommodation and bursaries will also be available.
The conference chairman is Sebastian Rahtz (Oxford University Computing Services) and local organisation is led by Kim Roberts (Oxford University Press).

Dates and Contacts

15th January 2000	Proposals for papers
31st January 2000	Acceptance of papers
15th February 2000	Publication of booking form and prices
31st March 2000	Delivery of papers for refereeing
31st May 2000	Delivery of final papers
General enquiries:	`tug2000-enquiries@tug.org`
Paper submissions:	`tug2000-papers@tug.org`

Sebastian Rahtz
OUCS
13 Banbury Road
Oxford OX2 6NN, UK
Tel: +44 1865 283431
http://tug2000.tug.org/

Toulouse, France **10–12 May 2000**

GUTenberg 2000 TOULOUSE

LaTeX and XML: Cooperating with the Internet

Toulouse has developped a reputation of being a dynamic city: aeronautics, space, electronics, and computers are its keywords. But even more, Toulouse lets you see its history as easily as its new modernity.

GUTenberg, the French-speaking TeX Users Group, also aims to be in the forefront of expanding frontiers. The upcoming GUTenberg meeting will bring together people who are working on LaTeX and XML developments evolving at an incredible rate right now on the Internet.

And so GUTenberg has chosen to hold its last meeting of the millennium in the Rose-Coloured City, from May 10 to 12, in the year 2000.

Authors are invited to submit their proposals in either French or English for consideration by the Program Committee. The first page should include the title, name and address of the author(s), as well as a time estimate for presentation.

Possible topics (not exhaustive)

- behind-the-scenes presence of LaTeX in browsers or other XML programs
- practical XML support for end-users (e.g., for Internet exchange by authors)
- a world-wide XML standard for the next decade
- XML use outside the Internet
- XML, search engines, browsers and public domain applications
- tools, editors, previewers, printer drivers for TeX engines
- LaTeX extensions
- world-wide archives, CTAN servers, maintenance, checking, improvements
- multi-language versions of tools, formats, documents
- fonts
- standardisation
- applications for: PostScript, PDF, SGML, HTML, XML, MathML, etc.
- graphics, sound, pictures
- editorial process
- aspects of scientific publication: tools for math, physics, chemistry, etc.
- LaTeX and competing products

Schedule

20 January 2000	submission of proposals
27 January 2000	acceptance notification
20 February 2000	deposit of paper
5 March 2000	submission of final paper
10–12 May 2000	conference in Toulouse

Ftp submission information

```
server: ftp.irisa.fr   password: toulouse
user:   gut2000        cd incoming
```

Create a directory using author name; deposit files, then send a message to `michele.jouhet@cern.ch` with details of directory and file names.

Upon acceptance, the necessary style files will be available from this server to produce the article in a form suitable for publication in the subsequent proceedings.

For information contact

Michèle Jouhet (President):
`michele.jouhet@cern.ch`
Bernard Gaulle:
`gaulle@idris.fr`
Anne Collin (GUTenberg Office):
`secretariat.gutenberg@ens.fr`

Further details to be posted at: `http://www.gutenberg.eu.org/manif/`

Participants at the 20th Annual TUG Meeting August 15–19, 1999, Vancouver, British Columbia

TUG '99 Attendees

Donald Arseneau
asnd@triumf.ca
Canada

Kiren Bahm
office@tug.org
USA

Frederick Bartlett
fredb@springer-ny.com
USA

Kaveh Bazargan
kaveh@focal.demon.co.uk
United Kingdom

Nelson H.F. Beebe
beebe@math.utah.edu
USA

Barbara Beeton
bnb@ams.org
USA

David Carlisle
davidc@nag.co.uk
United Kingdom

Lance Carnes
lcarnes@pctex.com
USA

Michael Carter
m.carter@econ.canterbury.ac.nz
New Zealand

Jae Choon Cha
jccha@knot.kaist.ac.kr
Korean

Daniel Christiansen
christiansen@albion.edu
USA

Kaja Christiansen
kaja@diami.au.dk
Denmark

Dennis Claudio
awkster@aol.com
USA

Helen C. Claudio
argon@its.caltech.edu
USA

Scott Collins
collins@siam.org
USA

Donald W. DeLand
deland@integretechpub.com
USA

Susan DeMeritt
sue@ccrwest.org
USA

Richard Detwiler
office@tug.org
USA

Simon Dickey
dickey@siam.org
USA

Michael Doob
mdoob@ccu.umanitoba.ca
Canada

Jean-luc Doumont
jl@jlconsulting.be
Belgium

Michael J. Downes
mjd@ams.org
USA

Gina Doxsey
texsales@pctex.com
USA

Peter Flynn
pflynn@imbolc.ucc.ie
Ireland

Erik Frambach
e.h.m.frambach@eco.rug.nl
The Netherlands

Yukitoshi Fujimura
yukif@ca2.so-net.ne.jp
Japan

Stephen Albert Fulling
fulling@math.tamu.edu
USA

Julian Gilbey
jdg@debian.org
USA

Steve Grathwohl
grath@math.duke.edu
USA

Peter M. Guinta
pgiunta@mit.edu
USA

Eitan M. Gurari
gurari@cis.ohio-state.edu
USA

Barbara Hamilton
hamilton@ccr-princeton.org
USA

Joseph Hertzlinger
jhertzli@ix.netcom.com
USA

Anita Z. Hoover
anita@zebra.us.udel.edu
USA

Patrick D.F. Ion
ion@ams.org
USA

Calvin Jackson
calvin@pcmp.caltech.edu
USA

Mimi Jett
jett@us.ibm.com
USA

Bob Johnson
bobj@synopsys.com
USA

Judy Johnson
jannejohnson@yahoo.com
USA

Tom Kacvinsky
tjk@ams.org
USA

N.G. Kalivas
ngk9131@westminster.org.uk
United Kingdom

Debra Kaufman
dkj@duke.edu
USA

Evelyn Kidd
evelyn.kidd@nrc.ca
Canada

Richard J. Kinch
kinch@truetex.com
USA

Ki Hyoung Ko
knot@knot.kaist.ac.kr
Korea

Siep Kroonenberg
siepo@cybercomm.nl
The Netherlands

Robert L. Kruse
bob@pretex.com
Canada

Warren Leach
warren@bluesky.com
USA

Silvio Levy
levy@math.berkeley.edu
USA

Douglas Lovell
dcl@watson.ibm.com
USA

Alex Lowrie
alowrie@educaide.com
USA

Pierre MacKay
mackay@cs.washington.edu
USA

Paul A. Mailhot
paul@pretex.com
Canada

Irina A. Makhovaya
irina@mir.msk.su
Russia

Jeffrey McArthur
jmcarth@atlis.com
USA

Denise McCall
denise@ccr-p.ida.org
USA

Wendy McKay
wgm@cds.caltech.edu
USA

Lothar Meyer-Lerbs
TeXSatz@uni-bremen.de
Germany

Bruce Miller
bruce.miller@nist.gov
USA

Frank Mittelbach
frank.mittelbach@latex-project.org
Germany

Mikael Möller
texab@faksimil.se
Sweden

Nadia Molozian
nadia_molozian@harcourt.com
United Kingdom

Patricia Monohon
pmonohon@zimm.ucsf.edu
USA

André Montpetit
montpetit@crm.umontreal.ca
Canada

Ross Moore
ross@mpce.mq.edu.au
Australia

Uwe Münch
muench@ph-cip.Uni-Koeln.de
USA

Timothy Murphy
tim@maths.tcd.ie
Ireland

Timothy Null
tsnull@worldnet.att.net
USA

Arthur Ogawa
ogawa@teleport.com
USA

Harry Payne
payne@stsci.edu
USA

Craig Platt
platt@cc.umanitoba.ca
Canada

Cheryl Ponchin
cheryl@ccr-p.ida.org
USA

Fabrice Popineau
fabrice.popineau@supelec.fr
France

Mike Potter
pottmi@lidp.com
USA

K. David Prince
kdp@u.washington.edu
USA

Sebastian Rahtz
sebastian.rahtz@oucs.oxford.ac.uk
United Kingdom

Heidi Rhodes Sestrich
heidi@stat.cmu.edu
USA

Nora Rogers
nora@scipp.ucsc.edu
USA

Chris Rowley
c.a.rowley@open.ac.uk
United Kingdom

Volker R.W. Schaa
v.r.w.schaa@gsi.de
Germany

Friedhelm Sowa
tex@sowa.rz.uni-duesseldorf.de
Germany

Donald P. Story
dpstory@uakron.edu
USA

Christina Thiele
cthiele@ccs.carleton.ca
Canada

Debbie Vose
vosede@lidp.com
USA

Hu Wang
hwang@aip.org
USA

Joseph Weening
jweening@ccrwest.org
USA

Alan Wetmore
awetmore@arl.mil
USA

De-Wei Yin
yin@asc.on.ca
Canada

Calendar

1999

Oct 7 – 10 ATypI '99, Association Typographique Internationale, Boston, Massachusetts. For information, visit `http://www.atypi.org/`.

Nov 3 – Dec 17 ABeCeDarium: A traveling juried exhibition of contemporary artists' alphabet books by members of the Guild of Book Workers, appearing at the Denison Library, Scripps College, Claremont, California. Sites and dates are listed at `http://palimpsest.stanford.edu/byorg/gbw`.

Nov 11 NTG 24th Meeting, Technische Universiteit Delft, The Netherlands. For information, visit `http://www.ntg.nl/`.

Dec 6 – 9 XML 99, Philadelphia, Pennsylvania. For information, visit `http://www.gca.org/conf/conf1996.htm`.

Dec 11 NTS talk by Hans Hagen, Masaryk University, Brno, Czech Republic.

Dec 13 Tutorial, "All the nice things we can do with pdf(TEX)", Hans Hagen, Masaryk University, Brno, Czech Republic. To attend, register with `secretary@cstug.cz`.

2000

Feb 7 *TUGboat* **21** (1), deadline for technical submissions.

Feb 7 – 11 Seybold Seminars Boston/ Publishing 2000, Boston, Massachusetts. For information, visit `http://www.seyboldseminars.com/Events`.

Feb 21 *TUGboat* **21** (1), deadline for reports and news items.

Feb 27 – Mar 2 XTECH 2000, the XML Developers Conference: "Looking back, going forwadr", San Jose, California. For information, visit `http://www.gca.org/attend/2000_conferences/xtech_2000/`.

Mar 8 – 11 DANTE 2000 and 22nd meeting, Technische Universität Clausthal-Zellerfeld, Germany. For information, visit `http://dante2000.itm.tu-clausthal.de/`.

Apr DK-TUG conference (proposed), Aarhus Universitet, Aarhus, Denmark. For information, visit `http://sunsite.auc.dk/dk-tug/`.

Apr 11 *TUGboat* **21** (2), deadline for technical submissions.

Apr 30 – May 3 BachoTEX 2000, 8th annual meeting of the Polish TEX Users' Group (GUST), "TEX on the turn of the 20th century", Bachotek, Brodnica Lake District, Poland. For information, visit `http://www.gust.org.pl/`.

May 9 *TUGboat* **21** (2), deadline for reports and news items.

May 10 – 12 GUTenberg 2000, "LATEX et XML : coopération pour l'internet", Toulouse, France. For information, visit `http://www.gutenberg.eu.org/gut/manif/gut99/`.

Jun 1 – 3 Society for Scholarly Publishing, 22nd annual meeting, Baltimore, Maryland. For information, visit `http://www.sspnet.org`.

Jun 12 – 16 XML Europe 2000, Palais des Congrès de Paris, France. For information, visit `http://www.gca.org/attend/2000_conferences/europe_2000/`.

Jun 16 – 18 TypeCon 2000, Westborough, Massachusetts. For information, visit `http://tjup.truman.edu/sota/`.

Status as of 1 November 1999

For additional information on TUG-sponsored events listed above, contact the TUG office (+1 503 223-9994, fax: +1 503 223-3960, e-mail: `office@tug.org`). For events sponsored by other organizations, please use the contact address provided.

Additional type-related events and news items are listed in the Sans Serif Web pages, at `http://www.quixote.com/serif/sans`.

Jun 21 – 23	TypoMedia 2000, "Future of Communication", Mainz, Germany. Linotype's design conference; for information, visit `http://www.typomedia.com`.
Jul 21 – 25	ALLC-ACH 2000: Joint International Conference of the Association for Literary and Linguistic Computing, and Association for Computers and the Humanities, Glasgow, Scotland, UK. For information, visit `http://www.ach.org/`.
Jul 23 – 28	SIGGRAPH 2000, New Orleans, Louisiana. For information, visit `http://www.siggraph.org/calendar/`.
Aug 12 – 18	**TUG 2000** — The 21st annual meeting of the TEX Users Group, "TEX enters a new millennium", Wadham College, Oxford, UK. For information, visit `http://tug2000.tug.org/`.
Aug 28 – Sep 1	Seybold San Francisco, San Francisco, California. For information, visit `http://www.seyboldseminars.com/Events`.
Sep	DK-TUG, 2nd Annual General Meeting. For information, visit `http://sunsite.auc.dk/dk-tug/`.
Sep 12	*TUGboat* **21** (3), deadline for reports and news items.
Sep 13 – 15	DDEP 2000: Digital Documents and Electronic Publishing, Munich, Germany. For information, visit `http://www.irisa.fr/ep98`.
Sep 19	*TUGboat* **21** (4), deadline for technical submissions.
Oct 17	*TUGboat* **21** (4), deadline for reports and news items.
Nov 17 – 19	Conference: Eric Gill & St. Dominic's Press, University of Notre Dame, Notre Dame, Indiana; three concurrent exhibitions of Gill's and related work will be held in the University museums and library. For information, visit `http://www.nd.edu/~jsherman/gill/`.
Dec 3 – 7	XML 2000/Markup Technologies 2000, Washington, DC. For information, visit `http://www.gca.org/attend/att_nxt_yrs.htm`.

Cartoon

by Roy Preston

TEX USERS GROUP

Promoting the use of TEX throughout the world

mailing address:
P.O. Box 2311
Portland, OR 97208–2311 USA

shipping address:
1466 NW Naito Parkway,
Suite 3141
Portland, OR 97209–2820 USA

Phone: +1 503 223–9994
Fax: +1 503 223–3960
Email: office@tug.org
WWW: www.tug.org

2000 TUG Membership Form

Rates for TUG membership and TUGboat subscription are listed below. Please check the appropriate boxes and mail payment (in US dollars, drawn on a United States bank) along with a copy of this form. If paying by credit card, you may fax the completed form to the number at left.

• 2000 TUGboat includes Volume 21, nos. 1–4.

• 2000 CD-ROMs include TEX Live 5 (1 disk) and Dante's CTAN (3 disk set).

• *Multi-year orders:* You may use this year's rate to pay for more than one year of membership.

• Orders received after March 1, 2000: please add $10 to cover the additional expense of shipping back numbers of TUGboat and CD-ROMs.

		Rate	Amount
Annual membership for 2000 (TUGboat and CD-ROMs)	☐	$65	____
Student/Senior membership for 2000 (TUGboat and CD-ROMs) (please attach photocopy of 2000 student/senior ID)	☐	$35	____
Subscription for 2000 (TUGboat and CD-ROMs) (Non-voting)	☐	$75	____
Shipping charge if after March 1, 2000.	☐	$10	____
Materials for 1999† (TUGboat Volume 20, TEX Live 4, 1999 CTAN CD-ROMs)	☐	$75	____
Voluntary donations			
General TUG contribution	☐		____
Contribution to Bursary Fund*	☐		____
		Total $	____

Payment (check one) ☐ Payment enclosed ☐ Charge Visa/Mastercard/AmEx

Account Number: _____

Exp. date: _____ Signature: _____

*The Bursary Fund provides financial assistance to members who otherwise would be unable to attend the TUG Annual Meeting.

† If you are a new TUG member wishing to receive TEX Live and CTAN right away, please order this item along with your 2000 membership.

We use the information you provide to mail you products, publications, notices, and (for voting members) official ballots, or in a printed or electronic membership list, available to TUG members only.

Note: TUG neither sells its membership list nor provides it to anyone outside of its own membership.

If you do *not* wish to have your name or other information in our membership list, please check here: ☐ .

Name: _____

Department: _____

Institution: _____

Address: _____

_____

_____

Phone: _____ Fax: _____

Email address: _____

Position: _____ Affiliation: _____