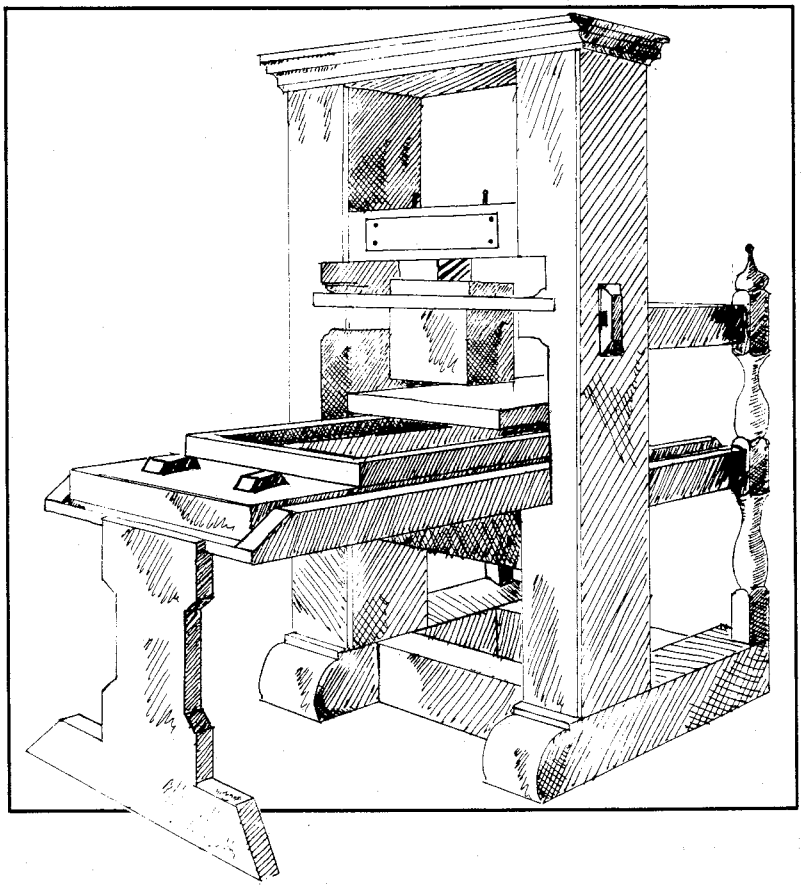
TUGBOAT

The Communications of the TEX Users Group



Volume 13, Number 4, December 1992

TEX Users Group

Memberships and Subscriptions

TUGboat (ISSN 0896-3207) is published four times a year plus one supplement by the TEX Users Group. As of December 1, 1992, the TUG office is moving from 653 North Main Street, P. O. Box 9506, Providence, RI 02940, U.S.A., to Balboa Building, Room 307, 735 State Street, Santa Barbara, CA 93101, U.S.A.

1993 dues for individual members are as follows:

- Ordinary members: $60
- Students: $30

Membership in the TEX Users Group is for the calendar year, and includes all issues of *TUGboat* and *TEX and TUG News* for the year in which membership begins or is renewed. Individual membership is open only to named individuals, and carries with it such rights and responsibilities as voting in the annual election. A membership form is provided on page 000.

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

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

Institutional Membership

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

Board of Directors

Addresses

General correspondence:
TEX Users Group
P. O. Box 869
Santa Barbara, CA 93102

Payments:
TEX Users Group
P. O. Box 21041
Santa Barbara,
CA 93121-1041

Parcel post,
delivery services:
TEX Users Group
Balboa Building
Room 307
735 State Street
Santa Barbara, CA 93101

Telephone

805-899-4673

Fax

[not known at press time]

Electronic Mail

(Internet)
General correspondence:
TUG@Math.AMS.com
Submissions to *TUGboat*:
TUGboat@Math.AMS.com

I do not know when the term "fine art" was invented and the breach between it and craftsmanship began to widen, but I have come to believe that it was a sorry day for both.

T. M. Cleland
"Progress" in the Graphic Arts
(1949)

TUGBOAT

COMMUNICATIONS OF THE TEX USERS GROUP

EDITOR BARBARA BEETON

VOLUME 13, NUMBER 4 • DECEMBER 1992
PROVIDENCE • RHODE ISLAND • U.S.A.

TUGboat

During 1993, the communications of the TEX Users Group will be published in four issues. One issue (Vol. 14, No. 3) will contain the Proceedings of the 1993 TUG Annual Meeting.

TUGboat is distributed as a benefit of membership to all members.

Submissions to *TUGboat* are reviewed by volunteers and checked by the Editor before publication. However, the authors are still assumed to be the experts. Questions regarding content or accuracy should therefore be directed to the authors, with an information copy to the Editor.

Submitting Items for Publication

The next regular issue will be Vol. 14, No. 1; deadlines for that issue will have passed by the time this issue is mailed. Deadlines for Vol. 14, No. 2 are February 16, 1993, for technical items, and March 16, 1993, for reports and similar items. Mailing dates for these two issues are scheduled for March and May. Deadlines for future issues are listed in the Calendar, page 530.

Manuscripts should be submitted to a member of the *TUGboat* Editorial Board. Articles of general interest, those not covered by any of the editorial departments listed, and all items submitted on magnetic media or as camera-ready copy should be addressed to the Editor, Barbara Beeton (see address on p. 415).

Contributions in electronic form are encouraged, via electronic mail, on magnetic tape or diskette, or transferred directly to the American Mathematical Society's computer; contributions in the form of camera copy are also accepted. The *TUGboat* "style files", for use with either `plain` TEX or LATEX, are available "on all good archives". For authors who have no access to a network, they will be sent on request; please specify which is preferred. For instructions, write or call the TUG office.

An address has been set up on the AMS computer for receipt of contributions sent via electronic mail: `TUGboat@Math.AMS.com` on the Internet.

Reviewers

Additional reviewers are needed, to assist in checking new articles for completeness, accuracy, and presentation. Volunteers are invited to submit their names and interests for consideration; write to `TUGboat@Math.AMS.com` or to the Editor, Barbara Beeton (see address on p. 415).

TUGboat Editorial Board

See page 415 for addresses.

Other TUG Publications

TUG publishes the series *TEXniques*, in which have appeared reference materials and user manuals for macro packages and TEX-related software, as well as the Proceedings of the 1987 and 1988 Annual Meetings. Other publications on TEXnical subjects also appear from time to time.

TUG is interested in considering additional manuscripts for publication. These might include manuals, instructional materials, documentation, or works on any other topic that might be useful to the TEX community in general. Provision can be made for including macro packages or software in computer-readable form. If you have any such items or know of any that you would like considered for publication, send the information to the attention of the Publications Committee in care of the TUG office.

TUGboat Advertising and Mailing Lists

For information about advertising rates, publication schedules or the purchase of TUG mailing lists, write or call the TUG office.

Trademarks

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

APS μ5 is a trademark of Autologic, Inc.
DOS and MS/DOS are trademarks of MicroSoft Corporation
METAFONT is a trademark of Addison-Wesley Inc.
PC TEX is a registered trademark of Personal TEX, Inc.
PostScript is a trademark of Adobe Systems, Inc.
TEX and $\mathcal{A}\mathcal{M}\mathcal{S}$-TEX are trademarks of the American Mathematical Society.
Textures is a trademark of Blue Sky Research.
UNIX is a registered trademark of UNIX Systems Laboratories, Inc.

General Delivery

The back benches beckon

Malcolm Clark

It might be appropriate to use this last opportunity to write as the President of the group to review the past year and produce a 'state of the nation' report. Rather than do this, I will simply present a brief mosaic of themes which reflect some of my current concerns.

I have always held the view that the TEX Users Group is an international group. A cursory glance at the membership list reveals many members in many countries. And this is not just because of the use of english as the *lingua franca* of the scientific and scholarly world. It is also because TUG is at the heart of a wide range of TEX-related activities (as usual, I use TEX as a shorthand for all the other bits and pieces which cluster round TEX). Therefore to site the next annual meeting in Britain (a small European offshore island between continental America and continental Europe) seems most appropriate. It simultaneously reflects the international nature of TEX and TUG, while acknowledging the great strides made within Europe in the formation of a large number of user groups, whose combined membership might rival TUG's international membership.

But there is more. It is especially noticeable that a sizeable amount of TEX activity revolves around electronic communication. True, the majority of TUG members and TEX users do not have straightforward (far less 'free') electronic access, but the activity generated by the privileged minority probably represents the bulk of development work going on at present. Aston University will be the host of the TUG meeting. Aston provides the largest repository of TEX material: it aggressively tracks down and installs style files, macro packages, public domain implementations for a wide variety of platforms, and all the other paraphernalia of the TEX-wise. Its recent announcement of a gopher service should simplify use of the archive. Just another good reason for choosing Aston.

The ease and facility of listservers and other forms of electronic communication sometimes fools us into believing that it is enough to create a list and subscribe to it. Since this ignores the majority of TUG members (and TEX users) we must be very wary of the conclusions and any decisions which develop. It has been an essential part of the LATEX3 project to inform the rest of the world of their progress and current thinking. This places an additional burden on the key personnel in the project, but since LATEX users will be among the beneficiaries of the project, they must be kept informed. The same sort of argument must be true for any other projects which may have far reaching effects on the TEX using community.

Each year brings a few new user groups: the national and language-based groups tend to attract most interest and attention, but we should not forget the many small 'local' groups which are around. This summer I had the pleasure of meeting many of the Santa Barbara User Group. Stimulated by meeting these enthusiastic and able people, Chris Rowley and I are seeking to initiate the West Hampstead TEX group here in London. There is surely scope for many more! At a rather larger scale, I recently attended a conference in L'viv in Ukraine which ended by recommending that a Ukrainian User Group be founded. TUG is commited to assisting groups at all scales, but it is clear that there is no one model of assistance: it may range from the purely symbolic to the practical and substantial.

By the time this is published, the new Executive Director will have been appointed, and my successor will be known. On these two people hinges a great deal. But not on them solely. An individual has only a small influence on TUG. All the membership influences the way and direction in which we are headed. Don't expect either of these two to be a magic bullet which will solve all ills (nor will future successes be theirs alone). We all have a part to play. I am acutely conscious that the stability and smooth(er) development over the last year or so has been aided conspicuously by a number of people. While it is invidious to single out individuals, I will do so nevertheless. I was ably supported by the other members of the Executive Committee, Bill Woolf, Ken Dreyhaupt and Peter Flynn; I am keenly aware of the support given myself, the Executive Committee, the Board, and the membership by Ron Whitney, the retiring Business Manager. These four people helped to make my TUG life so much easier and productive. I am confident that a similar relationship will develop in the future. And naturally there are many other individuals, some on the Board, some who I have met at conferences, and some with whom I have corresponded by email, who have all gone on to make my life as President of TUG rather interesting. I'm reminded of an announcement in

The Times where someone wished all of his friends and colleagues a Merry Christmas, 'except one'. But I could hardly be so uncharitable, or wish to be.

◇ Malcolm Clark
Information Resource Services
Polytechnic of Central London
115 New Cavendish Street
London W1M 8JS, England
UK
Janet: malcolmc@uk.ac.pcl.sun

Editorial Comments

Barbara Beeton

TEX: the past ...

From time to time, bits of trivia related to TEX's history come to light. The most recent is a copy of a letter that Don Knuth found in his files and forwarded to me with the comment that it was "written 2.5 years before I began working on TEX!!"

Dated November 7, 1974, it is addressed to Dr. Daniel Shanks of the Naval Ship Research and Development Center, Bethesda, Maryland; Shanks was at that time a member of the editorial board of the journal *Mathematics of Computation*. The letter states, in part,

> I would like to record my great disappointment in the quality of the new typography in *Mathematics of Computation*. I know that the change was caused by economic concerns, but I don't understand why we can't achieve in the 1970's what was routinely done in the 1870's. The type font is unattractive; the spacing between letters of a word is jerky and not conducive to smooth reading; a lot of the letters look slightly too large or too small. Although the right margin is ragged (and I don't mind this especially), many of the words are noticeably crowded together as if some margin alignment is being done anyway. This unattractive appearance will certainly discourage me from submitting any further papers to *MOC*, at least until all the other journals have deteriorated to the same level.

The October 1974 issue of *Math. of Comp.* was the first set in "cold type" — by a sophisticated typewriter. That method continued in use up to the first issue of 1981, when a switch was made to a composition system (not yet TEX) running on an in-house computer at AMS. The next piece of Don's writing published by the AMS was "Mathematical Typography",* the lecture that introduced TEX to the world at large.

Some more glimpses of the past appear on the following pages, in the transcription of a conversation between Don and Roswitha Graham, president of the Nordic TEX Users Group.

... and the future

Although Don announced in these pages that his work on TEX is complete, many users are concerned that there are things that TEX cannot do, that are nonetheless desirable and consistent with the practice of fine typography. Discussions are proceeding in several electronic discussion lists and at meetings of TEX users whenever they occur. The importance of this topic is such that the column dedicated to "future" topics, Dreamboat, has been moved from its former location near the end of *TUGboat* issues to a more prominent location immediately following this introductory section.

Two articles in this column deal with the state of TEX and possibilities for the future. The first, by Dick Palais (whom old-timers will remember as the founding chairman of TUG), gives a perspective colored by tradition and personal acquaintance with Don and the Stanford TEX project. The other is by Phil Taylor, who, while by no means a TEX newcomer, was introduced to TEX far from its point of origin and approaches the matter from quite a different direction.

Both Dick and Phil are faithful to Don's exhortation to create "masterpieces of the publishing art". If their methods differ, it is because their experience differs; I have never met two TEX users who have learned it in the same way, or even the same parts of it, and it has long since ceased to surprise me when I learn something new about TEX from even a new practitioner. The discussion is interesting, and Phil's article includes instructions on how to listen in or join it.

Another article (p. 510) solicits volunteers for tasks associated with the implementation of LATEX3. This important project, when complete, should provide a TEX environment even more attractive to new users than the present LATEX, as well as flexible methods for implementing the requirements of book and journal designers, features much desired by

* *Bulletin Amer. Math. Soc.* (N.S.) **1** (March 1979), 337–372

anyone working in a production environment. Your participation is encouraged.

TUG: the present

TUG is going through some changes.

Sometime very soon after I write this, the TUG office will be moving from Providence to Santa Barbara. A new Executive Director, Pat Monohon, will be taking charge. For much of the past year, Pat has been in charge of a group of volunteers who have been copying and distributing the public domain TEX packages that are available from TUG. So she has already begun to become familiar with some of the functions that are part of the TUG office duties. We wish her well.

I would also like to take this opportunity to recognize everyone who has worked so diligently in the Providence TUG office: Karen Butler, Cliff Alper, Teresa Pires, Kathy Sheely. They have been unstintingly helpful whenever I've had questions, and I shall miss working with them. And I'm not forgetting Ron Whitney — his contribution to making my job easier has been greater than I can say; in addition to managing the office, he has continued to respond to my requests for assistance with the *TUGboat* styles, and the credit for their reliability and ease of use in production is mainly his. Thank you all!

Not only is the office undergoing a transition — the elected management will be changing too. As already announced in *TEX and TUG News*, no one stepped forward to stand for election as TUG president, so the Executive Committee has studied the Bylaws for guidance in this situation. There *will* be a new president for a term beginning on January 1; the specifics will be announced in the next issue of *TTN*.

Please remember that this is your organization, and its success depends on you. Let's all of us, every TUG member, pull together and give our new representatives the support they deserve.

New ideas in typography

Earlier this fall I attended the Goudy Award Symposium at the Rochester Institute of Technology. This annual event, named for the esteemed American typographer, honors an outstanding type practitioner with an award and a program of talks about type and typographers.

One of the speakers this year was Peter Karow, of URW, Hamburg. His talk described a new typesetting program — the hz-Programm (named for Hermann Zapf, who designed the fonts on which it depends) — which optimizes text in such a way as to achieve nearly equal word spacing throughout each paragraph, using several interesting techniques that I believe are of interest to readers of *TUGboat*.

The first technique should be familiar already: paragraph-wide line breaking; this is, of course, the technique used in TEX, and though Karow didn't mention it during his talk, when I asked if the source were Knuth's algorithm his answer was "Of course!"

The second technique depends on specially designed fonts to lengthen or shorten lines to approach the target measure. For selected letters multiple shapes of differing widths are provided; typically these are letters that occur relatively frequently (e.g. "e") or have shapes that can by a small change have a significant effect on line length (e.g. "m").

A third technique can be called "intelligent kerning"; this consists in applying only positive kerning to lines that are shorter than the target length, and only negative kerning to lines that are longer.

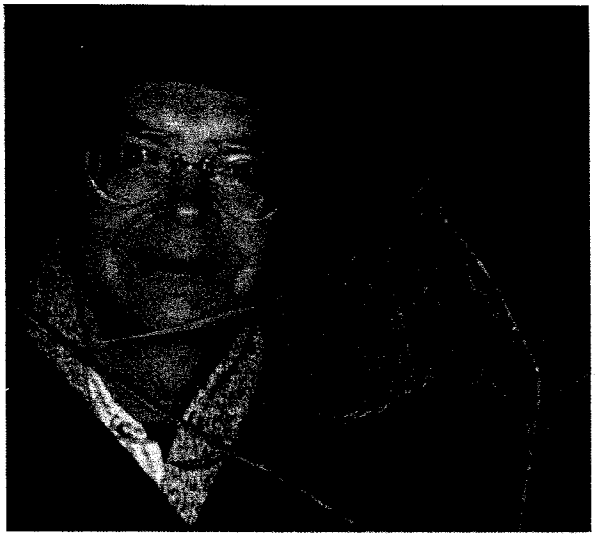
There are two more components to the package. The typographic quality of the resulting text is most impressive. The paper will be published elsewhere, but I have asked, and expect to receive permission, to reprint it here in a future issue.

◇ Barbara Beeton
American Mathematical Society
P. O. Box 6248
Providence, RI 02940
USA
bnb@Math.AMS.com

An Interview with Donald Knuth

In November 1991, Donald Knuth was one of a select group honored by appointments to Honorary Doctorates by the Royal Institute of Technology (KTH), Stockholm (see *TUGboat* 13, no. 2, p. 134). After his installation, he participated in a meeting of the Nordic TEX Users Group, where he responded to questions from the audience. He also spent some time talking informally with Roswitha Graham, the president of the Nordic group. These discussions were recorded, and edited transcriptions appear here with the permission of the participants.

Don also spent some time relaxing at the Grahams' country house on an island in the archipelago outside the Stockholm harbor. A helpful spider provided an appropriate setting for the author of WEB.



Question and answer session at the Nordic group meeting

Donald Knuth (DEK): I would like to say a big "Thank you" to Roswitha, and to the School of Computer Science and Department of Mathematics, for making my visit here possible and arranging everything. Also, I'm glad to be back here—I wanted to say that TEX owes a lot to the Nordic countries and to Sweden in particular.

I guess I never mentioned this in print, but when I designed TEX, I chose three examples of mathematical typesetting that I considered as standards of excellence, and I studied those three very carefully. I scanned them digitally—in those days we used a TV camera—and made a lot of careful measurements. One of the three was a volume of *Acta Mathematica*, printed in Stockholm about 1910. The second, in case you're wondering, was from the Netherlands in 1950, the mathematical section of their Academy of Science proceedings; the third was Addison-Wesley's house style as used in *The Art of Computer Programming*, when math composition was still done by hand.

Also, here at KTH you had one of the first four Alphatype machines running my original software. I received a copy of an early publication—I don't remember the name of it now, but I remember it had a green cover and it was a nice booklet about 40 pages long—a mathematical report. [It was *Non-linear Inverse Problems* by Gerd Ericksson (1983), 46 pp.; see "TEX incunabula," *TUGboat* 5, no. 1 (May 1984), 4–11.—ed.] So some of the very first extensive uses of TEX happened right at this institution. That makes me especially pleased to be here.

At the end of my courses at Stanford, I usually reserve the last day of class for a session we call "All questions answered," and I volunteer to answer questions on any subject whatsoever except religion or politics. And in recent years I've also excluded questions about Volume 4 of *The Art of Computer Programming*. [laughter] But today this looks like a very friendly audience, and I don't even mind if you have questions about Volume 4 of *The Art of Computer Programming*. So ask any questions whatsoever, go ahead. (Including politics.)

Question: I have a question about Chinese characters used with TEX.

DEK: Well, I don't know too much, but I'll tell you what I do know. One of the first scientific visitors from the People's Republic of China to the United States was Dong YunMei, who came to work with me and designed a system called LCCD, Language for Chinese Character Definition; he wrote a Stanford Computer Science report about that work. It was inspired by METAFONT, but he had special graphical primitives in there so that LCCD could do things that were especially important in Chinese for positioning radicals and so on. It was a language that wasn't based on, but was similar to METAFONT; he implemented it entirely himself. In the second edition of Volume 2 of *The Art of Computer Programming* (which was published in 1981, and was the first real use of TEX78), we used LCCD to typeset the Chinese names in the index. Then Dong went back to China, and now he heads a group at the Institute for Software Technology in Beijing. That group has, of course, done a lot toward extending his work since those early days. They now have a language called SP for typesetting Chinese and arbitrary Western texts; it was mentioned in the proceedings of the IFIP Congress, 1989, held in San Francisco; there's a brief, 5–6 page report about what is going on in computer research in China, and this is one of the projects mentioned. When he was at Stanford, he spelled his name "Tung"; now he spells it "Dong", but it's still pronounced "Doong". He's a very fine man. The most exciting thing for me is

that he's now directing a big project to use literate programming for Chinese, so they have a good way to document their programs, like the WEB system for documentation. This system, called CDP, combines the C programming language with the SP typesetting language. They also have an ongoing project for drawing Chinese characters electronically with a kind of metalanguage. However, I don't think they're anywhere near doing all the characters that way. That part of his group's activity seems to be more of a research project than a real production project.

There was another visitor from China at the same time, 1980—Mr. Ma, who worked at the Mathematics Department of the University of Beijing. He had some pretty good Chinese fonts at that time, which we could use on our laser printers. But that's all old stuff.

Just three weeks ago, I had a chance to meet Mr. Gu Guoan again. He's from Shanghai and now working for Ikarus, so he is affiliated with Peter Karow in Hamburg. Gu is the head of the Ikarus bureau in Shanghai. He originally came to Stanford from the Shanghai Printing Company, which is the largest printing company in China, shortly after Dong and Ma had returned to China. Gu wrote a paper with John Hobby that is mentioned in *The METAFONTbook*, describing about a METAFONT system for Chinese characters. That paper [*TUGboat* 5, no. 2 (1984), 119–136—ed.] was published as a Stanford Report and has lots of illustrations. They designed a system that did about a hundred test characters, all programmed in terms of 19 basic strokes. One of the basic strokes, for instance, is [draws on blackboard]. But they add a lot of parameters to it, so that it can appear in many different proportions. Then they have another basic stroke that looks something like this [more drawing]; you can have it tipped in different ways, and you can control exactly how much of a teardrop you have, and so on. So you have nineteen of these basic strokes, all done with parameters. Then you design the hundred Chinese characters by saying "do stroke number 1 in this position, and stroke number 2 in another position." The almost incredible thing about their system was that you could redesign the nineteen primitive subroutines and get completely different styles, still looking like real typefaces; so you could get Chinese in bold style, and you could get Chinese in the Ming Dynasty style, from the exact same descriptions of the hundred characters, just by changing the nineteen subroutines. Their article presented a kind of sans serif as well as a seriffed version, and they also had what they call a "Long" style or something like that. So they were getting three styles, and the design was a true meta-font in that sense. Their work was quite exciting to me, but as far as I know no one has pursued it; it was done by university researchers, who assume that when they're done, industry will march in and take over, but that didn't happen. I don't think there were any flaws in the research; I think it's just a matter of somebody picking up the idea and matched it with the right group.

On the other hand, Gu is now working for Ikarus, and is about to finish a massive conversion of Chinese characters into Ikarus format, then into PostScript with "hints"; this represents the entire set of all known Chinese characters—more than 50,000 but less than 60,000. He was visiting Adobe, so I happened to meet him a month ago when he came to California. His Ikarus descriptions won't be a meta-font, since they represent only one style. But he has access to the very best fonts, because the Shanghai Printing Company has these in the form of photographic masters.

– – * – –

Conversation with Roswitha Graham

Roswitha Graham (RG): Was anything new brought to you at the Nordic meeting, anything that you didn't expect to be there, as you listened to Frank [Mittelbach] and Yannis [Haralambous]?

DEK: As I said earlier, I was very impressed by the quality of the work that they're doing. I knew Frank's work before, because I'd talked to him at length when he came to Stanford. He had very penetrating insights into the whole program and has found many of the most subtle bugs in TEX. Also I know that he has this great commitment to quality publishing where he really wants to go a level beyond what I had ever conceived that I could do. So the proposals that he's made for going into even finer-than-fine quality printing, I know he comes from the right spirit. He might discover, as I did, that when you make some things better, other things get worse, and so finally you have to find a balance. But he might find a way to make everything better. So it's very inspiring to me to see the time that he puts into this and the vision that he has for making further advances. Also, an international perspective is important—people with different backgrounds from my own can obviously contribute things to TEX that I couldn't do myself; I'm only one person.

Then I met Yannis for the first time. His work..., that was the biggest surprise for me, to see how much he has achieved singlehandedly, doing things that I thought would need half a dozen or ten people to do, because of his background in many languages. Again, he has this great love for quality and love for preserving the beauty of "the good old days" in printing (where sometimes modern progress has taken steps backward). It was certainly exciting for me to see the care that he is taking to make keyboarding of so many kinds of text reasonable. When I was working on TEX, I wouldn't have dared to dream that it would become so easy to use the system in connection with Hebrew with vowel points, as well as Greek with all the breathings, and Ethiopian and Urdu and Arabic and many of these other things.

RG: Will you try this when you come home?

DEK: Well no, my life is too short. There are so many things that I could love very much spending my life doing—I could spend a lifetime doing fonts; I could spend a lifetime playing with color; I could spend a lifetime doing many languages. But I've already decided that the rest of my life is going to be *The Art of Computer Programming*. That's at least a twenty-year job, so I can't take any time off to do anything else from now on.

RG: I meant actually if you are going to try, not if you are going to work with it... just play with it?

DEK: I usually allow myself one or two days a year to play with some toy, so it's quite possible that I'll do that, one of these years.

RG: Did you get some mail from Beijing? We just got mail from Beijing, what this Chinese fellow had done. He had done some preprocessing.

DEK: Yes, that's thrilling for me too. People all around the world feel the same need for quality publishing that I had for mathematics publishing. So when I see people putting a lot of spare time into the work, I know that this is rewarding to them, and to many people afterwards having the fruit of their work. That's why it's exciting to me, of course, to have helped make this possible. I now see many publications where the authors were just inspired to create a fine book that they would never have begun if they hadn't had the power to do it themselves, and they needed something like TEX or METAFONT in order to be able to do it at all.

RG: Could TEX be an ideal program to communicate between different languages? Do you think it could be used in that way as well?

DEK: Well, as far as I know it's the best existing system to control things like paragraphing while dealing with many languages with respect to hyphenation and so on. But certainly I never thought that TEX would be the last word. In fact, one thing hasn't happened that I was expecting: When people were faced with special kinds of publishing projects, I was assuming that they would just make their own version of TEX instead of trying to use TEX as it is. For example, if someone wanted to come up with a new edition of the Bible, I was really expecting them to use some WEB change files, hiring a programmer who would be able to give them new features adapted to their project. You know, if you are going to produce an Arabic/English dictionary, or something like that, you'll want special capabilities. I never expected TEX itself to be a general tool for all these purposes. Now I find that people are close enough to being able to do most of these things, that hardly anybody is actually changing the WEB code. Instead, they've found that they could do almost everything they want with standard TEX. Yannis found a way... I said, are you worried that I have only 256 characters in the font when you're dealing with Greek? (The first fonts of Greek that were developed in Europe, in the Netherlands, had some six hundred characters including a huge number of ligatures.) But he assured me that he's not at all worried about this restriction. He doesn't find the existing system problematical.

Now I see here the report from Chinese TEX, and it's got chemical formulas as well as Chinese and mathematics, done with a preprocessor. Of course, to do some of these things you can't expect macros would handle everything the easiest way. My goal was to build an engine that would be versatile, that would be very good at balancing texts and able to do hyphenation in a variety of languages, and so on, and I was hoping that it would facilitate printing the languages of people all over the world—and now here it is!

Since Professor Romanovskii is with us today, I'm reminded also of recent developments in Russia. We all know that when people do some work they're proud of, they also want to make it look like something they're proud of. In America, I couldn't write a book that I thought was going to be produced by a printer who wouldn't do it well. In the Soviet Union, when it was government policy not to have easy access to Xerox machines or laser printers, the policy was holding everybody back from being able to communicate their best work. So it's also a big hope of mine that a new

... freedom isn't the right word for it, I guess I mean an ability to communicate to the world community in mathematics ... will lead to a great flourishing of new Russian books. So much Soviet mathematics I've seen had to be condensed into a few pages. I'm sure the authors were just as proud of their results as we are — deservedly even more proud — but just had no way to let it out. It's the same with computer programs as with mathematics. Computer programs need to be communicated, to be read by people.

The communication factor is the real, to me the most important reason why so many changes began to happen in the Soviet Union. The rest of the world was improving its communications rapidly, while in the Soviet Union, computers were used mostly for numerical calculations and couldn't be used extensively for word processing; it was too much against the policy of the government. So this need for communication, also for beautiful communication, is something that I had long wished for in every part of the world, but especially in the Soviet Union. Jill can tell you about times when I would play Tchaikovsky on the piano, with tears in my eyes because I was thinking how beautiful this music is, but how I wished that my friends in Russia could be doing some of the other things that I was doing.

Communication between machines has become the real reason for computers in most of the world. China faces the same problem; they can never reach modern levels of technology until all limitations are removed from their communication networks.

RG: What difference is there between the Japanese way of doing TEX and the Chinese way?

DEK: I'm not familiar with it, but in both cases they seem to be getting quality. So I'm happy, as long as the output looks great.

I get completely unexpected mail every once in a while. Did I tell you the story about Italy? I guess not. *The TEXbook* has been translated not only into Japanese, but also into Italian (though not yet published in Italian). The people who did the translation are actually prisoners, political prisoners thrown into jail in the '70's because they were liberal activists. But they're in a minimum security prison, so they're supposed to work for their living. This one group decided that their job was going to be to typeset mathematics with TEX for the Italian Mathematical Society; so they did this. They sent me a letter explaining how they're prisoners, and they enclosed a laser-printed copy of the entire *TEXbook* translated into Italian, 500 pages of it, a beautiful job. They claim they are enjoying their prison life because of TEX. What could be nicer?

RG: That's a point of view I never thought of, actually. I find it great that you have given to the world something that is free of charge and it's a challenge for everyone who wants to communicate. If the possibility is given, I think it's wonderful there is something like this. I find it amazing that so many people have put in so many working hours without ever asking any labor union about it.

DEK: It was part of my original thinking that TEX should not be competing with a labor union, or even with other people's egos. In other words, I knew that if I had been working at Bell Laboratories, for example, I could never do anything that would compete with `troff`, because that would be not respectful to the other people there who had put years and years into it. To me, `troff` had proved that the whole idea of something like TEX was possible, and therefore it was a good idea to start over from scratch and think over how to do it if you were starting over. Well, it's impossible to do such things in an existing organization. Similarly, if I hadn't done TEX for mathematics, if I had had as my first goal to do newspapers, or the yellow pages, or something that counts for the majority of publications in the world, then (if successful) I would have been putting a lot of other people out of work and making them angry at me. I didn't want to do that. So I was happy to be creating a tool for mathematics; it seemed to me that nobody in the world enjoyed typesetting mathematics. Good — I wasn't going to offend anybody. This was a strong component of my thinking as I wrote TEX. Well, it turned out there was one person who was offended, and he complained at one of the TUG meetings; this man from Science Typographers had spent a lot of his time making a commercial system, which is still used. He keeps improving it, and it's excellent. There's absolutely no reason to question the quality of the journals his company typesets, in any way. But he did complain, and that made me feel bad, because I didn't think I was going to make anybody unhappy by doing TEX.

Another thrill at this Nordic meeting was to learn that TEX will soon be used to typeset the journal *Acta Mathematica*. The main reason it's exciting to me is that a circle is being closed: *Acta Mathematica* was one of my first models for quality when I designed TEX. As I said at the meeting, I had selected three standards of excellence that I did a lot of measurements on, hoping to be able to

match their quality. *Acta Mathematica*, in 1910, was the mathematics journal that had the best budget in the world at the time, and they did a very good job; so I had looked at it very closely in 1977. Now comes the surprise: Right after the TEX meeting, I ran into Jan Michael [Rynning] at the Institute talking to people about how to do *Acta Mathematica* with TEX! The decision has apparently been made now. Not only that, but Leif Andersson showed at the Nordic TUG meeting that he understood all the intricacies of making new mathematics fonts for TEX. So it should be possible to do *Acta Mathematica* in the fonts that you had in 1910 as well.

RG: Did they agree to?

DEK: Who knows, but he could certainly do it. The expertise exists in Sweden to make this possible.

I tried to design TEX so that people who would be spending a lot of time working with it would still find their job pleasurable. Suppose they have to deal with TEX many hours of every day for years and years. I tried to make it so that they would still find this a pleasant job, because they could keep discovering new constructions that would be somehow fun or beautiful or satisfying, in amongst all the other things they have to fight in order to cope with unusual constructions that always come up in printing. My hope was that people could continue to find yet more elegant ways to solve certain problems, and enjoy the process. The number of people around the world who are putting in this extra effort seems to indicate that the language is working in this respect as I had hoped.

RG: Just being interested in people communicating ..., because I am visualizing that it will be possible to have this text in a mail and it could even be translated by the language.

DEK: Oh yes, we could build it into fax machines and so on. I hope that in January I will see the absolute last bug in TEX, and somebody can make a chip so that it will be easy to have TEX inside any machine!

The American Math Society is making TEX source available now; the *Math Reviews* are available in TEX form going back fifteen years or so, and they continue to extend this so that mathematicians can look for an article on a certain topic and can see it typeset on the screen. And this is something that could certainly work for all sorts of complicated applications.

But you still need, if you are to get the highest quality, to have copy editors review everything and make sure you have the right style of quotation marks, for example. Every academic discipline has its own problems and every standard reference work can use typography in ways that make the reference work more effective but also is specific to that book. So every book is a new challenge. The advantage of TEX is that it's able to be adapted to these different challenges; it doesn't put everything into a single format. On the other hand you could also consider that a disadvantage.

Suppose you were allowed to rewrite all the world's literature; should you try to put it all into the same format? I doubt it. I tend to think such unification is a dream that's not going to work.

Authors have adapted themselves to the medium that they are using at the time, and if we're going to understand the authors' intentions, we should see something close to that medium. The medium of the future may be some other sort of standard for international communication; whatever the future brings, people can use that method. Authors will express themselves best in the medium they are using for creation. Authors using TEX now can, for example, write a different kind of mathematical paper than they did ten years ago, because they know they can choose a notation that isn't going to be misunderstood by a typesetter who knows no mathematics. So I can write a more effective technical paper if I know that I can make the diagrams correct—I can put in different kinds of diagrams and tables than I would have dared to if I was going through other people who didn't understand my intentions. Things that were written 20 years ago were written in a style that was optimized for older technology. Movies are analogous: People who made silent films chose different scenes than they would in a sound film, but their work was effective as a silent movie. And you can't just take a movie that's black and white and colorize it and get the same effect. It's the same thing with older texts as well as texts for special purposes and other languages. So this idea of a standard format is only something for the future, not for the past.

RG: You can't leave the field! You'll follow up in some way what's happening with TEX, even if you now are going to devote your time to something else, you cannot let it go. You have spent so much time... Is it a hate/love, or is it only love?

DEK: Well, it's hate when I learn that I made yet another error. I just got a message in the mail saying that if somebody sets the math unit to a

negative value — which nobody in their right mind would ever do, but if they do — apparently TEX goes crazy. So I have to fix that. Ugh.

What I love is when excellent new publications come out that I know wouldn't have been done at all without TEX, and also when I see people — as I said, Frank and Yannis — spending a considerable part of their lives doing work that has very high quality. They are excited just by the chance of improving the quality of publication. Those are the things that make me happy.

RG: Thank you very much.

Dreamboat

Moving a Fixed Point

Richard Palais

Abstract

In the past few years there has been increasing discussion of the question "Has the time has come to make basic changes to the inner workings of TEX?". In late May of 1992, Rainer Schoepf set up a mailing list on the Internet, called "NTS-L", to discuss the matter. I started out being completely opposed to the idea of even the slightest changes to the TEX code, feeling that whatever failings TEX might have, they are best approached by pre and post processing ("front and back ends"), and anyway are negligible compared to the danger of losing the remarkable coherence and interchangeability of TEX software, everywhere and on all platforms that is enforced by the discipline of having a single, universally accepted underlying piece of software (INITEX). However, after following the discussion carefully for nearly two months, I was convinced by evidence that, for certain purposes, TEX was no longer fulfilling its promise of providing typesetting of uncompromising high quality, and probably only careful and limited changes and additions to TEX primitives could correct this. What follows is a long message I posted to NTS-L, outlining a minimalist approach to changing TEX, and also a suggested method for implementing changes to TEX code that would insure documents written for standard TEX could still run under the new system. A number of replies to my message were posted to NTS-L and others were addressed to me personally via email. Rather than incorporate these comments by making appropriate changes to the version I posted, I have decided to append a short addendum, mentioning a few of the more important points made in these replies.

– – * – –

This is going to be more a "position paper" than a simple message. I have been following the NTS-L mail list discussion with considerable interest and finally felt that there were so many issues that I wanted to address and remarks that I wanted either to agree with or to dispute, that only a fairly extensive reply would do. Here is a table of contents:

Introduction

First a short personal introduction. The oldtimers of the TEX world will perhaps remember me — I was the founding chairman of TUG, worked closely with Don Knuth during the early years of TEX, and I wrote a column on mathematical typesetting in the *Notices of the AMS* for three years, with the goal of easing the transition in the mathematical community from the typewriter, along WYSIWYG road, and into the bright new Promised Land of TEX. But my name may well be unfamiliar to more recent arrivals in the TEX world, for lately I have been only a "lurker" on comp.text.tex, and while I read *TUGboat* and use TEX daily for writing my letters, papers, and books, and in connection with my duties as an editor of the *Bulletin of the AMS*, I have not recently been contributing either to the development or to the public discussion of TEX.

Next a disclaimer. While I know my way around in *The TEXbook* and have been writing my own macros and formats since 1978, I consider myself an amateur, not at all in the same league with TEXperts like Barbara Beeton, Michael Downes, Victor Eijkhout, Karl Berry, Larry Siebenmann, Tim Murphy, and others who have been contributing to this discussion. So I will happily defer to them on technical matters and hope that they will correct any of my misstatements. What I would like to do is take the point of view of a devoted TEX

user; one not so enamoured of TEX as to be unable to see its warts, but one who appreciates what a unique software miracle TEX is, and is willing to try to fix things only if assured that it will not subvert that miracle. One more fact about me bears emphasizing; as a mathematician I do have a somewhat biased view of TEX. For me TEX is not just *a* typesetting system, it is *the* mathematical and "TEXnical" typesetting system.

I would like to begin with a quotation from Don Knuth's "Remarks to Celebrate the Publication of *Computers & Typesetting*" at the Computer Museum, Boston, Massachusetts, May 21, 1986, as printed in *TUGboat*, vol. 7 (1986) no. 2, pp. 95–98:

> ...Ever since these beginnings in 1977, the TEX research project that I embarked on was driven by two major goals. The first goal was *quality*: we wanted to produce documents that were not just nice, but actually the best...My goal was to take the last step and go all the way to the finest quality that had ever been achieved in printed documents...
>
> The second major design goal was to be *archival*: to create systems that would be independent of changes in printing technology as much as possible. When the next generation of printing devices came along, I wanted to be able to retain the same quality already achieved, instead of having to solve all the problems anew. I wanted to design something that would still be usable in 100 years. In other words, my goal was to arrange things so that, if book specifications are saved now, our descendants should be able to produce an equivalent book in the year 2086. Although I expect that there will be a continual development of "front ends" to TEX and METAFONT, as well as a continual development of "back ends" or device drivers that operate on the output of the systems, I designed TEX and METAFONT themselves so they will not have to change at all: They should be *fixed points* in the middle, solid enough to build and rely on.

Perhaps it is because I was in the audience when Don made those remarks that they seem particularly important to me, but in any case, as my contribution to the NTS discussion, let me attempt to analyse the TEX system and some of its purported shortcomings in the light of Knuth's quotation. More specifically, I would like to address the following:

QUESTIONS.

1) *Are Knuth's two goals consistent, or has the continual quest for ultimate quality in typesetting exposed problems with TEX so intractible that they cannot be addressed simply by creating new and better front and back ends for the TEX system?*
2) *If so, can these "intractible" problems be solved by changes to TEX that will leave it compatible with the current version (and in particular able to pass Knuth's "trip-test").*

The Many Faces of TEX

TEX is a complex system that can appear as many things to different people (or even to one person at different times). In fact it is a little like the proverbial elephant that the blind men perceived in so many ways depending on how they "interfaced" with it.

I think that this many-faceted nature of TEX may account, at least in part, for some of the unfocused and chaotic discourse that has been taking place on this mailing list. Someone will comment either critically or in praise of one aspect of the TEX system and someone else will contradict that comment, but really in reference to some other aspect of the system. As anyone scanning comp.text.tex realizes, LATEX users face a whole different set of problems than plain TEX users, and likewise $\mathcal{AMS}$-TEX and LA$\mathcal{MS}$-TEX provide still other environments, with differing attendant strengths, problems, and difficulties. The complaint, repeated several times in the recent discussions, that TEX is incompetent to do commutative diagrams, may seem obvious to a frustrated user of plain TEX, but it would perplex a user of LA$\mathcal{MS}$-TEX who will tell you that it is an absolute snap using "TEX" to make beautiful commutative diagrams, even very complicated ones with arrows set at almost arbitrary slopes and with all kinds of decorations on them. Likewise, it is well-known that designing tables can be a painful chore with (plain) TEX. But there are a number of excellent macro packages around that automate this problem away. Even that most serious problem of integrating graphics into TEX can be considered solved in the right TEX environment. In the hands of a competent artist, a Macintosh equipped with Textures, Adobe Illustrator, and a PostScript printer can create strikingly professional integrated graphics and text. Yes, I know that this solution gives up the portability of TEX documents—bad things can sometimes even happen between the proofing device and the high resolution camera copy typesetter—but the point is that *many apparent problems with TEX can be solved by coupling TEX to suitable front and back ends, with no reprogramming at all of TEX itself.* Someone suggested that TEX

needs Bézier curves as a new primitive. I will argue that the Bézier curves belong in an Illustrator-like program, *not* in TEX. Solving the problem of portability is trivial in comparison with the nightmarish difficulties that I foresee as virtually certain to follow from trying to add anything so foreign as Bézier curves to TEX's data structures!

A "Standards" Approach to Solving TEX Portability Problems

As just suggested above, I believe that at least some of the major defects currently perceived in the TEX system are not so much problems with TEX itself, but rather arise from the vital requirement that *TEX documents should be completely portable between various hardware platforms.* As long as we are dealing with TEX itself, this portability is assured by the minimal requirement that all true TEX systems will produce the same DVI file from a given source file. But of course a DVI file is only part of the way to a printed page, so TEX without some sort of back end is virtually useless. We sometimes forget that even the software combinations formed by a set of font glyphs (either bitmaps or outlines) and a screen previewer or printer driver is already a back end to TEX. If we are willing to stick with the Computer Modern family of fonts in the bitmapped format provided by METAFONT, then virtually all screen previewers and printer drivers will work faultlessly and provide "identical" output to a tolerance limited only by resolution. The reason of course is that these fonts are a carefully specified standard, on which the writer of a device driver can completely rely. But of course Knuth never intended TEX to be limited to the CM family of fonts, or even to METAFONT designed fonts. Currently, Adobe's PostScript Type 1 fonts are the world's favorite, and it has become increasingly the case that a typesetting system, if it is to remain acceptable, **must** be able to deal at the very least with the basic thirty-five fonts built into PostScript printers. Of course TEX was easily up to the challenge. All that is necessary is to build a TFM file for each Type 1 font (or better yet an AFM to TFM conversion program), and add the basic code to the device driver to handle a Type 1 font. On any given system this is an easy task, since again the Type 1 format is a completely specified standard. I know this was done several years ago on the Macintosh, and I believe it has also been done for most of the other major hardware platforms. There are now even a number of well hinted Type 1 versions of the basic Computer Modern fonts available. However even this quite simple new back end leads to portability problems between systems. I have never tried it, but I suspect strongly that if I sent a colleague with an IBM clone one of my Textures source files that used Times Roman, it would not work under PCTEX or emTEX without modification. The problems here are quite trivial, involving little more than differences in font naming conventions. All that would be necessary to regain complete cross-platform portability when using PostScript fonts is some standardized naming conventions. I have made a point of this not because it is a difficult problem that has worried people much; rather because it is a simple problem with an easy solution — but one that I think can be generalized to solve many other TEX problems without in any way tampering with TEX itself.

For a hard example, let's consider a problem that has been the subject of a great deal of discussion in the TEX community and in *TUGboat*, namely specifying graphics within a TEX source file. Of course one possibility that has been mentioned would be to add a number of graphics primitives to TEX: lines, circles, Bézier curves, colors, fills, bitmaps, etc. To my mind this would be absolute madness, and I find it hard to believe any one would seriously consider it. The obvious reason to reject this approach is that it would lead to a program infinitely more complex than TEX that could never be made bug free or portable. Moreover in a few years, when Bézier curves are perhaps out of fashion, and some new graphics goodies are all the rage, there will be a call for yet another "upgrade" of TEX. But a better reason to reject it is that one should not attempt to brush one's teeth with a paintbrush or try to paint a picture with a toothbrush — use the correct tool for each job. And while Swiss Army knives may make fine souvenirs and conversation pieces, they are not high quality tools.

The simple and straightforward solution is to consider a graphic as just another box (a "bounding box"), just like any other TEX box, and let some appropriate back end worry about what is inside the box and render it appropriately on a screen or sheet of paper. Then one can always create graphics with the very best front end graphics tools currently available on a given platform, save it in an appropriate ASCII-based file format, such as encapsulated PostScript, tell TEX about its bounding box and its format, and let the back end take over from there. "But wait a minute," you say, "isn't that exactly the old "`\special`" approach?" Of course it is, and I claim that

the \special mechanism has worked very well *except* for the problems with portability that it has introduced. Now experience has taught that the correct approach to portability problems is *not* to create complex do-it-all programs and then struggle to make them work on dozens of different platforms. Rather, one should have single purpose modules with simple data structures and well-defined interfaces, and use these to build up more complex systems. So, I maintain that what is required to solve the portability difficulties caused by graphic elements in TEX is to make a serious effort to set up cross-platform TEX standards for various officially recognized graphics formats and a standard syntax for \specials to go along with them. It would have to be understood that as technology advances, older formats will probably die out and be replaced by newer ones, so there should probably be a standing committee, perhaps of TUG, to oversee the promulgation and maintenance of these graphics standards. In the same way there could be another standing committee for setting TEX standards for font formats and naming conventions for fonts.

By the way, while we are on the matter of fonts and standards, let me complain about what I feel is a serious failing of the TEX community. The Grand Wizard, as a sort of parting gift, gave us a potentially very valuable tool to handle all sorts of font problems. This was in the form of a well-defined standard — I'm referring of course to virtual fonts (VF). I'm a little over my head here technically, but I believe that as well as solving the more obvious problems for which they were introduced, virtual fonts could be used to handle some more esoteric tricks like adding color and other attributes to fonts. But my feeling is that we have dropped the ball. Not enough TEX systems have implemented VF to make it a dependable way to solve cross-platform TEX problems — even Blue Sky Research, which prides itself in providing a state of the art TEX environment for their Textures system on the Macintosh, has yet to implement it.

Let me end this part of the discussion with a mention of one thing that I feel should neither be a part of NTS nor even a standardized front end for it, and that is the user interface. I would not have brought this up except that there has been discussion on this list giving favorable mention to creating a standardized graphical user interface as part of NTS. But the hardest part of programming these days, and the most system dependent, is building a GUI. Even on a single platform, like the Macintosh, these can break when a new system update comes out. In general, even with systems as close in spirit as the Mac OS, Windows, and NeXT, it is extremely difficult to write a uniform GUI for a program meant to run on several platforms, and porting a GUI from one of these to say X-Windows on UNIX would be even harder. Moreover, each platform has certain User Interface Guidelines for its own GUI, and users get quite upset when a program deviates from them. Since these guidelines differ from one platform to the next, some users, and most likely all, would be upset by any uniform choice. Finally, what is the point? All this would do is stifle creativity and progress. Let the implementors of NTS on each platform design and construct the user interface most suitable for that platform.

The Matter of Compatibility

There has been a lot of discussion on NTS-L concerning the question of whether NTS should necessarily be compatible with the current version of TEX. Until this point I have tried to be calmly analytical, but this is a crucial issue, and one I feel very strongly about, so I am going to drop into a more polemical mode at this point (though I will try to keep my arguments rational). In a word I feel that *backwards compatibility is an absolute sine qua non for any system that aspires to be accepted as a "successor" to TEX.*

Of course, if a group wants to break off to design a completely new typesetting system from scratch that is fine with me — just as long as they don't use TEX in the name or pretend it is some sort of "successor" to TEX. As for me, I would like to see NTS be an improved version of TEX, and for this, it should either be 100% compatible with TEX, or if not it should at least default to a "compatibility mode" which is 100% compatible. I will suggest later a method by which major internal changes could be made to TEX and still satisfy this essential requirement, but now let me be precise about what I mean by compatibility and say why I feel that this a no-compromise issue.

INITEX is the core TEX program, the basic compiled version of the TEX code that knows only TEX's primitives. In a certain sense INITEX *is* TEX. It is the implementation of INITEX that determines whether a "TEX" system is authentic, i.e., passes Knuth's trip-test, and I think there is little doubt that INITEX is one of the "fixed points" that Don was referring to in the above quotation. Let me argue as strongly as I can that *whatever NTS is, its core typesetting function should be based on* INITEX — a version that will pass the trip-test. The

reason has nothing to do with "keeping the faith". Rather it is purely practical. If the new system is compatible with TEX, it will find ready acceptance. But if it is not, then the immense installed base of TEX users will almost certainly shun it, and it will consequently be stillborn.

Let me provide some details about the part of this "user base" that I know something about, the mathematical community, since I have seen comments on the mailing list that indicate a serious lack of comprehension of how sizable this group is (relative to the TEX community) and how dependent it has become on TEX. This in turn may have led to what I consider a very unfair comment, namely that TEX is a "toy for mathematicians". By the way, while my firsthand knowledge is restricted to mathematics, I know by hearsay that much of the following holds true for theoretical physics and also in many other scientific and technical disciplines in which mathematical text makes up a substantial part of papers written in that discipline.

First, virtually all mathematics graduate students now write their dissertations in TEX, and from then on write all their papers in TEX. Secondly, nearly all mathematicians below age forty have learned TEX, and an increasing number of the older generation are either switching to TEX, if they write their own papers, or else are having their secretaries and technical typists learn TEX and write their papers in it. A couple of years ago many mathematicians were still using WYSIWYG mathematical word processors, but now one sees very few preprints prepared in any format except TEX. There are of course lots of reasons for this rapid, wholesale switching to TEX, and probably different reasons have been important for different people. Here are a few:

- Mathematics set by TEX looks much more professional.
- Setting mathematics with TEX is faster and easier (after a painful, but short, learning curve).
- Mathematical text in TEX format can be sent over the Internet and works on all machines. This makes TEX an ideal medium for joint authors to use in their collaboration. WYSIWYG formats are machine dependent and need special coding and decoding when sent over the net.
- As a result of the above, the TEX mathematical input language is becoming a *lingua franca* for the linearization of mathematical text in email and other ASCII documents, even if they are not meant for typesetting.
- The two largest mathematical publishers, the American Mathematical Society and Springer-Verlag (and many others besides), now accept papers in TEX format, either on disc or over the Internet. Papers submitted this way often get published more rapidly and of course final proofreading is minimal.

In any case, the mathematical community now has become so dependent on TEX and has such a substantial investment in software, personal macro files, and source files for the current version of TEX, that I believe *it is virtually certain to reject any purported successor system that does not protect that investment.*

Since I seem to be at odds with Mike Dowling on this matter, let me quote some of his remarks and point out an important issue he seems to have overlooked:

> (1) Upwards compatibility is a very minor issue for the user. Theses are written only once; there is little or no need to recompile under the successor to TEX after the thesis has been submitted. The same comment goes for publications. It is easy to dream up exceptions to this, but I contend that they are just that, exceptions. (A good counter example is a script accompanying a course. This script will be modified and recompiled every time the course is offered.)

Well, let me dream up another minor exception for you! If you take a look in your local science library you will find several feet of shelf space occupied by the issues of Mathematical Reviews (MR) from just the past year. In fact, every year the American Mathematical Society not only publishes many tens of thousands of pages of books and primary mathematical journals in TEX, it also publishes more tens of thousands of pages of MR. The cost of producing just one year of MR is well in excess of five million dollars, and all of MR going back to 1959 (about one million records) is stored online *in TEX format* in the MathSci database. People all over the world download bibliographic data and reviews from MathSci and use TEX software to preview or print it. Many others spend hundreds of dollars per year to lease two CD-ROMs with the last ten years of MathSci. Obviously the AMS is unlikely to agree with the above assessment of the importance of compatibility. In fact they are certain to protect their investment in MathSci by making sure that the retrieval system they have invested in so heavily does not break. And they have a

powerful means to protect that investment — with Knuth's blessing, they own the trademark on the TEX name and logo, and will not let it be used for a system that does not pass the trip-test.

TEX as a Front End

Early in the NTS-L discussion there was some discussion concerning extending TEX so it could flow text around pictures, and have other sophisticated facilities of page layout programs such as PageMaker or QuarkXPress. This quickly died out, I think because most people on the list had thought enough about such matters to realize that typesetting and page layout are almost orthogonal activities. The ability of TEX to break text into lines, paragraphs, and pages is aimed at producing printed pages consisting mainly of text for books and journals. Of course, such pages frequently do need diagrams, pictures, and other graphic elements. But these usually fit neatly inside captioned boxes, with no need to have text flow around them, and we have already discussed making such extensions to TEX. The page layout programs, on the other hand, are designed with the quite different purpose of producing illustrated magazines, newsletters, and newspapers. These are documents in which the graphics often outweighs the text, and in which each page can have a complex, and different pattern of text and pictures. Building such pages is an interactive process best handled with a WYSIWYG interface. The good page layout programs often have only quite limited word-processing facilities built in, because the proper way to use them is *not* for creating either text or graphics, but rather to organize into pages text and graphics imported from other programs.

But this brings up an interesting point. To what extent would it be possible to import text typeset by TEX into a page layout program? Certainly this would not be easy! The way TEX freezes the shape of a paragraph, once it has created it, is quite different from the way a normal word processor works, so one would probably have to create a special page layout program, one that understood TEX's data structures and could have an interactive dialog with TEX during the layout process. This would be a tough but worthy undertaking.

TEX as a Programming Language

Many contributors to NTS-L have complained that the TEX programming language is terrible. In its favor one should point out that it is Turing effective — and so just as powerful as say C or Pascal — and it is the programmability provided by this macro language that gives TEX its remarkable flexibility and survivability. However, there is no denying that, while TEX macros may indeed always behave exactly the way (a careful reading of) the TEXbook says they will, it often takes a lot of study for a non-wizard to find the features responsible for a macro behaving the crazy way it does, rather than the way that was intended. Still, most TEX users do learn easily enough to write simple substitution macros or even special purpose macros with parameters. The real problems arise when one tries to write a complex package of general purpose macros for others to use in an unknown environment. One can take the attitude that this activity is simply intrinsically difficult, and should be left to the experts, but it seems to me that those complaining have a good point. Someone who has learned to program in a standard programming language should not have to learn another whole new system of programming; they should be able to use the familiar syntactic and semantic features that they are used to for programming TEX. Since changing the TEX macro language would introduce the worst kind of compatibility problems, some other solution is called for. One that comes to mind is to write a "compiler" whose source language would be some sort of high-level, ALGOL-like language, with all the usual features such as strongly typed variables and scoping rules, and whose target language would be the TEX macro language. Creating such a compiler would not be an easy task, but it would constitute another important application of Knuth's principle of keeping TEX itself a fixed point while making "changes" to the TEX system by creating new front ends.

Changing the Fixed Point

I would be a lot happier if I could stop at this point and conclude that there is no need for any changes to the TEX code itself — that all of TEX's perceived problems can be solved by creating the appropriate front and back ends. For the overwhelming majority of TEX users this is in fact the case. If one is willing to put up with occasionally having TEX fall just short of perfection, or if one doesn't mind making up for these lapses on TEX's part by doing some careful manual tuning (my own approach), then the current TEX is all one will ever need. But for those who take seriously Knuth's goal of not compromising on quality, and moreover insist on a system that permits them to automate excellence, a

very good case has been made that TEX has several serious deficiencies hard-wired into it.

Frank Mittelbach made this point very cogently and convincingly in his presentation "E-TEX: Guidelines for future TEX" at the 1990 TUG meeting (published in *TUGboat* vol. 11 no. 3, September 1990). And Michael Downes amplified and extended Mittelbach's comments in a message he sent to the tex-euro mail list, February 20, 1992, in response to an announcement by Joachim Lammarsch of the intent of Dante e.V. to set up a working group on "Future Developments of TEX". Downes posted a copy of that message to NTS-L on June 2, 1992, and I see no need to repeat either of their remarks here.

Instead I would like to suggest a mechanism to permit necessary changes to be made to TEX code and still maintain compatibility in the sense described above. The idea is both simple and obvious. When NTS starts up it will be ordinary TEX. However if the first string of characters in the source is, let us say, "`\VERSION=NTS`" then the TEX code will be rolled out of RAM and replaced with NTS code.

But how are we going to get from TEX to NTS? My own preference would be to take a gradual approach, analyzing the problems that have been pointed out in TEX into families of related problems, each reasonably independent of the others, and then tackling these families one by one in stages, from easiest to hardest, starting from the original TEX sources and gradually perturbing them. In this way NTS could evolve in a controlled way from the current version of TEX through a sequence of versions, each compatible with standard TEX, each new version curing one more of the difficulties that Mittelbach, Downes and others have pointed out, and each being carefully tested before going on to the next stage. I know this may seem like a dull and pedestrian way to go about things, particularly to those wishing to strike out boldly in new directions. But I think it has the a very good chance of success. It will not demand many resources to get started so it stands a reasonable chance of getting off the ground. And once the first step is taken, well as the saying goes, nothing succeeds like success.

Summary

Let me now summarize my major points and suggestions:

- Many of the problems and "missing features" in the TEX system that have been discussed in NTS-L are not really deficiencies of TEX, but rather features omitted as a consequence of Knuth's decision to limit the functionality of TEX, in order to make it stable and transportable. Many of these problems have been solved in a quite satisfactory manner on one or more platforms by coupling TEX with the appropriate front or back end. What remains is to solve these problems in a manner that preserves transportability of TEX sources, and the way to do this is to specify standard file formats and other data strucures, and a standard `\special` syntax for instructing TEX to interact with them.
- To carry out the above, TUG should appoint a "Committee on TEX Standards". This committee should have the overall responsibility for deciding what types of standards are important to insure that important front and back ends for TEX can be built in a way that is platform independent, and it should appoint committees of experts to promulgate and maintain these various standards.
- Nevertheless, an excellent case has been made that certain specific features of TEX's primitives and coding make it nearly impossible to automate certain functions required to attain one of Knuth's goals for TEX, production of "the finest quality that had ever been achieved in printed documents". While most users may never feel the need for the subtle touches that make the difference between typesetting that is merely excellent, and typesetting that is "the finest quality", for those that do a follow-on to TEX, NTS, should be developed.
- NTS should be backward compatible with source files from the current version of TEX. This means that it should default to a "compatibility mode" that would pass the trip-test, and that any new features that might introduce incompatibilities should have to be "turned on" by the user.
- NTS should be developed in a sequence of versions, starting with TEX and curing its problems one at a time.

Postscript

As indicated above, I believe it is possible for a group to design and implement *ab ovo* a completely new and state of the art typesetting system — a "TEX for the Twenty-first Century" to use Philip Taylor's words. As explained above, I also believe that such a system could be implemented in a way that would keep it functionally compatible with the current TEX system. But, before getting started on

such a massive project, ample consideration should first be given to some prior considerations:

- Don't forget what a monumental task the creation of TEX was, and remember that its author is a totally exceptional individual. He is not only a great computer scientist who happens to love and understand high quality typography, he is also, fortunately, an incredibly good programmer — and finally he has unmatched *Sitzfleisch*. Whole work groups of system analysts and programmers could easily have failed in the same task — and if they had succeeded they would probably have taken longer to create a buggy program that runs on a single platform. *And they certainly would not have put the code in the Public Domain!*
- Knuth is a tenured Full Professor at Stanford. While he was designing TEX and writing the code, he had NSF grant support that not only provided him with the time and equipment he needed, but also supported a team of devoted and brilliant graduate students who did an enormous amount of work helping design and write the large quantity of ancillary software needed to make the TEX system work.
- So, consider this question: Where will the resources come from for what will have to be at least an equally massive effort? And will the provider of those resources be willing, at the end of the project, to put the fruits of all this effort in the Public Domain? I consider this point particularly important. I think it is accepted that it is the combination of the quality and the PD status of the TEX code that have been the two principal factors responsible for its remarkable and unique universality. I doubt that any system that is not PD would have much chance of weaning away a sufficient number of TEX users to make all the effort worthwhile.
- Finally, don't repeat the sad history of ALGOL 68! The ALGOL 60 programming language was a gem. True, it had its flaws, but these were well-known and understood, and I think all of us ALGOL lovers assumed that the ALGOL 68 design committee was going to polish that gem for us and remove the flaws. Instead they decided to start over from scratch and came up with a language that nobody understood, loved, or used. And that spelled the doom of poor old ALGOL — who was going to maintain an ALGOL 60 compiler once ALGOL 68 was "on the way"? Needless to say, even a botched NTS isn't going to kill TEX, but it would be sad to waste all that time and effort — and a great opportunity.

Addendum

A number of people responded to my posting — some by email directly to me, and others by a posting of their own to NTS-L. I would like to thank all who took the trouble to reply, but for reasons of space I will mention here only a couple of replies that bear most directly on my previous remarks.

I would particularly like to thank Nelson Beebe for pointing out that several of the front and back ends I was wishing for either already exist or are in the works. First, and perhaps most important, Nelson himself has made a proposal for a standardized syntax for `\specials` that he has submitted to the TUG DVI Committee, and this will appear shortly in *TUGboat*. Second, Nelson reminded me of an article by Luigi Semenzato and Edward Wang in the November 1991 issue of *TUGboat*. This describes a LISP front end for TEX macro writing, of just the sort I was calling for in the section "TEX as a Programming Language". (But I'd still like to see one based on an ALGOL family syntax!) And finally he pointed out Graham Asher's article "Inside Type & Set" in the April 1992 issue of *TUGboat*, describing a program that does page makeup with lines and paragraphs typeset using TEX code.

Larry Siebenmann sent me a long list of interesting comments, however I will not mention them here since I hope and expect he will himself write something on these matters in these pages.

◇ Richard Palais
Department of Mathematics
Brandeis University
Waltham, Massachusetts 02254
palais@binah.cc.brandeis.edu

The Future of TEX*

Philip TAYLOR

This paper is dedicated to Professor Donald Knuth, without whom there would simply be no TEX, no METAFONT, *and almost no chance that any of us would ever have met, on the occasion of the approximate anniversary of his pronouncement two years ago that TEX and* METAFONT *were complete.*

Abstract

TEX and the other members of Knuth's *Computers & Typesetting* family are arguably amongst the most successful examples of computer software in the world, having been ported to almost every conceivable operating system and attracting an allegiance that verges on the fanatical. Development work on this family has now ceased, and many members of the computer typesetting community are concerned that some action should be taken to ensure that the ideas and philosophy enshrined in TEX are not allowed simply to fade away. In this paper, we discuss some of the options available for perpetuating the TEX philosophy, and examine the strengths and weaknesses of the present TEX system. We conclude by postulating a development strategy for the future which will honour both the letter and the spirit of Knuth's wish that TEX, METAFONT and the Computer Modern typefaces remain his sole responsibility, and at the same time ensure that the philosophy and paradigms which are the strengths of TEX are not lost for ever by having artificial constraints placed on their evolution.

– – * – –

"My work on developing TEX, METAFONT and Computer Modern has come to an end." [1] With these words, Professor Donald E. Knuth, creator of TEX, informed the world that the evolution of probably the most successful computer typesetting system yet developed had ceased, and that with the sole exception of essential bug fixes, no further changes would be made. TEX's version number will asymptotically approach π as bug fixes are made, and at the time of his death, it will be renamed 'TEX, Version π'; thereafter it will remain exactly as he last left it: a fitting and appropriate memorial to one of the most productive and inspired computer scientists (and mathematicians, and Bible scholars) that the world has ever known.

The future of TEX is therefore totally determined: why, then, is this paper entitled *The Future of TEX*? Because, primarily, TEX is already fifteen years old — four years as a child (TEX 78); eight years as an adult (TEX 82); and three years in maturity (TEX 3). Fifteen years is a long time in the lifespan of computer languages: Algol 68, for example, was certainly at or beyond its peak by 1982, and is today almost as rare as the Tasmanian wolf,[1] if not yet as dead as the Dodo:[2] a language must evolve, or die. (There are numerous natural languages which are almost certainly in terminal decline, despite the most strenuous efforts of a nucleus of active speakers to artificially prolong their lives: Cornish and Manx are surely dead; Gaelic must feature in any linguistic 'Red Book' of endangered languages; only Welsh, which alone among the British native minority tongues continues to evolve, shews any real resistance to morbidity and eventual death.) If natural languages must evolve or die, how much more so must computer languages, whose evolution must keep pace with a technology which evolves at a rate so rapid that it is unmatched in the natural world even by irradiated fruit-flies.[3]

So, my underlying hypothesis is: TEX must evolve, or die. If we are to believe the evidence of our ears and eyes, the underlying TEX philosophy is already as anachronistic as the horse and cart: TEX represents the pinnacle of Neanderthal evolution, building on the genetic heritage of Runoff, Nroff, Troff, Ditroff and Scribe, whilst Cro-Magnon man, in the guise of Ventura Publisher, Aldus Pagemaker and Quark Xpress, is already sweeping over the face of the planet. The halcyon days are long since gone (or so it would seem) when it was socially acceptable to: enter text; check it for spelling errors (by eye!); insert a series of formatting commands; pass the whole through an interpreter; identify the first error; correct the first error; pass the whole through the interpreter again; identify the second error; correct the second error; pass the whole through the interpreter for a third time; repeat for all subsequent errors...; pass the whole through the interpreter for the n^{th} time; then pass it through the interpreter again (to resolve forward- and cross-references); preview a facsimile of the final copy on the computer screen; notice a formatting error; and go right back to editing the file: our colleagues sit there clicking away on their mice[4] like demented death-watch

* This article is reproduced by kind permission of the organisers of the Euro-TEX '92 conference in Prague, Czechoslovakia, in the proceedings of which [4] it first appeared.

[1] *Thylacinus cynocephalus*

[2] *Raphus cucullatus*

[3] *Drosophila melanogaster*

[4] *Mus ordinatus microsoftiensis* or *Mus ordinatus applemacintoshii*

beetles[5] and think us totally mad; and mad we surely must be, for we not only enjoy this mode of working, we seek to convert the demented mouse clickers into TEX users as well!

Why? What is it about TEX that is so totally addictive? Is it perhaps TEX's descriptive and character-oriented nature — the fact that, in direct opposition to current trends, TEX requires the user to think about what he or she wants to achieve, and then to express that thought as a series of words and symbols in a file, rather than as a series of ephemeral mouse movements on a screen? Is it, perhaps, its portability — the fact that implementations (almost entirely public domain) exist for every major operating system in the world? Is it the deterministic nature of TEX — the fact that a given sequence of TEX commands and text-to-be-typeset will always produce *exactly* the same results, regardless of the machine on which it is processed? Is it the 'boxes and glue' paradigm, which provides a simple but somewhat naïve model of black and white space on the printed page? The ease with which form and content can be separated? The implementation as a macro, rather than a procedural, language? (would a procedural TEX still be recognisably TEX?) Is it, perhaps, the incredible contortions through which one occasionally has to go to achieve a desired result? (Or the incredible elation when such contortions finally achieve their intended effect?) How many of these elements could be eliminated and still leave something that is recognisably TEX? I propose to return to these questions, and to attempt to answer some of them, later in this paper.

A related question: what is the potential lifespan of a TEX-based typesetting system, or for that matter, of any computer language? Of all the general purpose computer languages which have sprung into existence since the advent of compilers (which point in time really marks the beginning of all the computer languages that are in general use today), Cobol and Fortran are probably among the longest lived; but Fortran has evolved enormously since the days of Fortran 2 (which is as far back as my memory goes), whilst Cobol has evolved relatively little; Basic, too, is still with us, although the originators of Dartmouth Basic would find little to recognise in the 'Visual Basic' of Microsoft today. Algol 60 evolved via various routes into Algol 68, which for me represents the pinnacle of language design, but evolved no further, and is today reaching the end of its twilight years. Pascal, which owes much to the Algol family, gave birth to Modula, which itself became transmuted into Oberon; in a sense, this last example represents a failure of the evolutionary system, for in its heyday Pascal was almost universally adopted, giving birth to the UCSD 'P' system as well as making possible the unbelievably successful (and revolutionary) 'Turbo Pascal', whilst Modula, although lauded by computer scientists, remained of relatively limited acceptance and acceptability, and Oberon remains almost unknown without the walls of academia. Most recently, among the procedural languages at least, we come to 'C', and its bastard offspring 'C++'; these languages have an honourable history, tracing their roots back through 'B' (or so I am told — I have never encountered 'B' myself) to BCPL, the 'Basic Combined (or Cambridge, depending on one's background) Programming Language', itself derived from CPL which simply wasn't so basic! *En route*, data typing was acquired, and lost, and acquired again, and polymorphism was acquired with the advent of 'C++'. Other evolutionary lines are represented by Prolog, which epitomises the declarative family, and Lisp, which is the archetype of list processing languages (and which remains almost unchanged since its inception). Poplog, encompassing as it does representatives of all three families (Pop 11, based on Pop 2, Prolog and Lisp) is perhaps a unique synthesis. Finally one should not omit mention of that most modestly titled of all programming languages, APL: '*A* Programming Language'.

But this is not a history of programming languages: I cite the above examples only to place TEX within context, for although when teaching TEX to secretaries one does not necessarily stress the fact of its being a computer programming language *per se*, a computer programming language it most certainly is. Indeed, TEX is 'Turing complete', which is a computer scientist's jargon for saying that TEX could be used as a general purpose programming language since it has the necessary flexibility, although apart from the intellectual satisfaction there would be little point in so doing: TEX's *forte* is clearly computer typesetting, and only programmers or perverts could derive pleasure from coercing it into calculating cube roots or cosines!

So what is the common theme among all the languages cited above? Simply this: that almost every one of them has either given birth to a successor (which is not necessarily more successful: *cf.* Pascal → Modula → Oberon), or has simply fallen into disuse; Cobol and Lisp alone, which occupy highly specialised niches, remain relatively unchanged, and of these only Cobol continues to play a significant rôle in mainstream computing (although Lisp remains the language of choice for many linguistic and related tasks).

[5] *Xestobium rufovillosum*

It seems, then, that we have a choice: we can either allow natural selection to take its course, in which case TEX, having fulfilled its appointed rôle on this planet (which I assume is to teach us the merits of literate programming, whilst encouraging us to devote ever more time to the typesetting of beautiful papers, presumably at the expense of ever less time spent actually researching or writing them), will surely join XCHLF, JEAN & JOSS in the great bit-bin in the sky; or we can adopt a corporate responsibility for the future of TEX and intercede in the process of natural selection, taking steps to ensure that TEX evolves into a typesetting system which is so demonstrably superior to the miasma of mouse-based, menu-driven, manipulators of text and images which are currently snapping at its heels that no-one will be able to deny it its rightful place at the forefront of typesetting technology for the twenty-first century.

Let us consider the options which are available to us:

1. We can leave TEX exactly as it is: this is clearly a defensible position as it is exactly what Knuth himself intends to do; it would be extremely arrogant of us to suggest that we know better than Knuth in this respect.
2. We can enhance TEX by just enough that those who really understand its power, its limitations, and its inner workings agree that it no longer has demonstrable defects (i.e. there are some 'simple' typesetting tasks with which TEX_π could not deal correctly, but with which an enhanced TEX could).
3. We can enhance TEX by incorporating the combined wish-lists of its major practitioners, thereby seeking to make TEX all things to all men (and all women), whilst retaining its present 'look and feel'.
4. We can enhance TEX as in option 3 above, whilst taking the opportunity to re-consider, and perhaps substantially change, its present look and feel.
5. We can take the opportunity to do what I believe Knuth himself might do, were he to consider today the problems of typesetting for the first time: look at the very best of today's typesetting systems (clearly including TEX among these), and then design a *new* typesetting system, far more than just a synthesis of all that is best today, which addresses the needs and potential not only of today's technology, but that of the foreseeable future as well. We would need to find some way to incorporate that spark of genius which characterizes Knuth's work!

No doubt each of us will have his or her own ideas on the desirability or otherwise of each of these options; it is not my intention in this paper to attempt to persuade you that any one of them is clearly preferable; but I would be shirking my responsibilities were I not to caution that, in my opinion, option 3 appears to represent the worst of all possible worlds, representing as it does a clear case of 'creeping featurism' at its worst while not possessing any redeeming qualities of originality.

Option 1 is, as I have suggested above, clearly defensible, in that it is Knuth's own preferred position; despite my fears that TEX will succumb to the pressures of natural selection if it is adopted, it may be that TEX represents both the pinnacle and the end of an evolutionary line, and that future typesetting systems will be based on an entirely different philosophy (e.g. mouse-based).

Option 2 represents the most conservative evolutionary position and has, I believe, much to commend it, certainly in the short term: it would retain the present look and feel of TEX; and compatibility with current TEX programs, whilst not intrinsically guaranteed, could be ensured by careful design; at the very worst, one could envisage a command-line qualifier which would disable the extensions, leaving a true TEX 3 underneath. Although option 2 is in opposition to Knuth's expressed wishes, he has made it plain that he has no objection to such enhancements *provided that* the resulting system is not called TEX. I propose that we term the results of adopting option 2 'Extended TEX', both to indicate its nature, and, more importantly, to comply with the spirit as well as the letter of Knuth's wishes.

Option 3 is considerably less conservative, but does at least retain the present look and feel of TEX; it is completely open-ended in terms of the extensions made to TEX, and offers the opportunity to make sweeping enhancements (I hesitate to use the word 'improvements' for the reasons outlined above). Compatibility with current TEX programs need not prove problematic, provided that the design were adequately thought out, and again the possibility of a '`/noextensions`' qualifier provides a fallback position. The timescale for such an implementation would not be small if a new swarm of bugs is to be prevented, and it is not clear how future obsolescence is to be avoided: after all, if 'The Ultimate TEX' (as I will term it) includes all the proposed enhancements of TEX's major practitioners, what enhancements remain to be implemented in the future?

Option 4 represents the first attempt at a true

re-design of TEX, allowing as it does the option to rethink TEX's look and feel, whilst continuing to incorporate many of its underlying algorithms. One could envisage, for example, an implementation of TEX in which text and markup were kept entirely separate, with a system of pointers from markup to text (and *vice versa*?). One advantage of such a scheme is that it would eliminate, at a stroke, the troublesome nature of the <space> character which currently complicates TEX; the escape character could become redundant, and the problems of category codes possibly eliminated. Of course, this is just one of many such possibilities: once one abandons the look and feel of TEX, the whole world becomes one's typesetting oyster. One might term such a version of TEX 'Future TEX'.

Option 5 is without doubt the most radical: not only does it reject (at least, initially), TEX's look and feel, it challenges the entire received wisdom of TEX and asks instead the fundamental question: "How should computer typesetting be carried out?" In so doing, I believe it best represents Knuth's own thoughts prior to his creation of TEX 78, and, by extrapolation, the thoughts which he might have today, were he faced for the first time with the problems of persuading a phototypesetter to produce results worthy of the texts which it is required to set. I think it important to note that there is nothing in option 5 which automatically implies the rejection of the TEX philosophy and paradigms: it may well be that, after adequate introspection, we will decide that TEX does, in fact, continue to represent the state of the typesetting art, and that we can do no better than either to leave it exactly as it is, or perhaps to extend it to a greater or lesser extent whilst retaining its basic model of the typesetting universe of discourse; on the other hand, neither does it imply that we *will* reach these conclusions. I will call such a system 'A New Typesetting System' (to differentiate it from 'The New Typesetting System' which is the remit of NTS, *q.v.*).

The options outlined above are not necessarily mutually exclusive: we might decide, for example, to adopt option 2 as an interim measure, whilst seeking the resources necessary to allow the adoption of option 5 as the preferred long-term position (indeed, I have considerable sympathy with this approach myself). But no matter which of the options we adopt, we also need to develop a plan of campaign, both to decide which of the options is the most preferable (or perhaps to adopt an option which I have not considered) and then to co-ordinate the implementation of the selected option or options.

As many of you will be aware, a start has already been made to this end: at a meeting of DANTE (the German-speaking TEX Users' Group) earlier this year, Joachim Lammarsch announced the formation of a steering group, organised under the ægis of DANTE, to co-ordinate developments of TEX; this group, diplomatically called 'NTS' so as to avoid any suggestion that it is TEX itself whose future is being considered, is chaired by Rainer Schöpf; the members are listed in Appendix . An e-mail discussion list has also been created (called NTS-L),[6] with an open membership;[7] all messages are automatically forwarded to members of the NTS team. At the time of writing this article, the group has not yet formally met: instead, we have been content to listen to the many positive suggestions which have been put via the medium of NTS-L. It is clear that there is no general consensus at the moment as to which of the five options outlined above is preferable; some argue for strict compatibility with existing TEX implementations, whilst others argue that we must grasp the nettle and take this opportunity to create a truly revolutionary typesetting system. Some, at least, are quite content to adopt the Knuthian position, and simply use TEX as it is: "TEX is perfect" was the subject of more than one submission to NTS-L. One of the more interesting facts to emerge from the discussion is the different ways in which TEX is perceived: some see it simply as a tool for mathematical typesetting; others want to be able to create the most complex graphics without ever leaving TEX's protective shell; many want to be able to typeset arbitrarily complex documents (not necessarily containing one line of mathematics), but are content to leave graphics, at least, without TEX's remit.

So far, this paper has been concerned primarily with generalities; but I propose now to look at some of the specific issues to which I have earlier merely alluded, and to offer some personal opinions on possible ways forward. I propose to start by attempting to answer the question which I believe lies at the very heart of our quest: "What is the essence of TEX?"

It seems to me that there are some aspects of TEX which are truly fundamental, and some which are merely peripheral: among the fundamental I include its descriptive and character-oriented nature, its portability, and its deterministic behaviour; I also include some elements which I have not so far discussed: its programmability

[6] NTS-L@VM.URZ.Uni-Heidelberg.De

[7] Send a message to
LISTSERV@VM.URZ.Uni-Heidelberg.De
with a single line body containing the text
Subscribe Nts-L <given name> <SURNAME>

(for example, the way in which loops can be implemented, even though they are not intrinsic to its design), its generality (the fact that it can be used to typeset text, mathematics, and even music), its device independence, and its sheer æsthetic excellence (the fact that, in reasonably skilled hands, it can produce results which are virtually indistinguishable from material set professionally using traditional techniques). Equally important, but from a different perspective, are the facts that it is totally documented in the ultimate exposition of literate programming (the *Computers & Typesetting* quintology), that it is virtually bug-free, that any bugs which do emerge from the woodwork are rapidly exterminated by its author, and finally that for higher-level problems (i.e. those which are at the programming/user-interface level rather than at the WEB level), there are literally thousands of skilled users to whom one can appeal for assistance. We should not forget, too, Knuth's altruism in making the entire source code[8] freely available with an absolute minimum of constraints. It is almost certainly true that this last fact, combined solely with the sheer excellence of TEX, is responsible for TEX's widespread adoption over so much of the face of our planet today.

Among its more peripheral attributes I include its implementation as a macro, rather than as a procedural or declarative, language, and perhaps more contentiously, its fundamental paradigm of 'boxes and glue'. I hesitate to claim that boxes and glue are not fundamental to TEX, since in many senses they clearly are: yet it seems to me that if a descendant of TEX were to have detailed knowledge of the *shape* of every glyph (rather than its bounding box, as at present), and if it were perhaps to be capable of typesetting things on a grid, rather than floating in space and separated by differentially stretchable and shrinkable white space, but were to retain all of the other attributes asserted above to be truly fundamental, then most would recognise it as a true descendant of TEX, rather than some mutated chimera.

Without consciously thinking about it, I have, of course, characterized TEX by its strengths rather than its weaknesses.[9] But if we are to intervene in the processes of natural selection, then it is essential that we are as familiar with TEX's weaknesses as with its strengths: if it had no weaknesses, then our intervention would be unnecessary, and the whole question of the future of TEX would never have arisen. But whilst it is (relatively) easy to identify a subset of its characteristics which the majority of its practitioners (I hesitate to say 'all') would agree represent its fundamental strengths, identifying a similar subset of its characteristics which represent its fundamental weaknesses is far more contentious. None the less, identify such a subset we must.

Perhaps the safest starting point is to consider the tacit design criteria which Knuth must have had in mind when he first conceived of TEX, and which remain an integral part of its functionality today. TEX, remember, was born in 1978—a time when computer memories were measured in kilobytes rather than megabytes, when laser printers were almost unknown, when the CPU power of even a University mainframe was probably less than that available on the desktops of each of its academics today, and when real-time preview was just a pipe dream.[10] Each and every one of these limitations must have played a part in TEX's design, even though Knuth may not have been consciously aware of the limitations at the time. (After all, we are only aware of the scarcity of laser printers in 1978 because of their ubiquity today; we aren't aware of the limiting effects of the scarcity of ion-beam hyperdrives because they haven't yet been invented....) But by careful reading of *The TEXbook* (and even more careful reading of `TEX.WEB`), we can start to become aware of some of the design constraints which were placed on Knuth (and hence on TEX) because of the limits of the then-current technology. For example, on page 110 one reads: "TEX uses a special method to find the optimum breakpoints for the lines in an entire paragraph, but it doesn't attempt to find the optimum breakpoints for the pages in an entire document. *The computer doesn't have enough high-speed memory capacity to remember the contents of several pages* [my stress], so TEX simply chooses each page break as best it can, by a process of 'local' rather than 'global' optimization." I think we can reasonably deduce from this that if memory had been as cheap and as readily available in 1978 as it is today, TEX's page-breaking algorithm may have been very different. Other possible limitations may be inferred from the list of numeric constants which appear on page 336, where, for example, the limit of 16 families for maths fonts is stated (a source

[8] including source for the TEX and METAFONT books; this is frequently forgotten...

[9] OK, I admit it: TEX *might* have weaknesses...

[10] Although on page 387 (page numbers all refer to *The TEXbook* unless otherwise stated), we find "Some implementations of TEX display the output as you are running".

of considerable difficulties for the designers of the New Font Selection Scheme);[11] 16 category codes, too, although seemingly just enough, force the caret character (^) to serve triple duty, introducing not only 64-byte offset characters and hexadecimal character specifiers, but also serving as the superscript operator.

So, we may reasonably infer that the combined restrictions of limited high-speed memory, inadequate CPU power, and very limited preview and proof facilities, combined to place limitations on the original design of TEX, limitations the effect of which may still be felt today. It is perhaps unfortunate that in at least one of these areas, that of high-speed memory, there are still systems being sold today which have fundamental deficiencies in that area: I refer, of course, to the countless MS/DOS-based systems (without doubt the most popular computer system ever invented) which continue to carry within them the design constraints of the original 8088/8086 processors. Because of the ubiquity of such systems, there have been a fair number of submissions to the NTS list urging that any development of TEX bear the constraints of these systems in mind; despite the fact that I too am primarily an MS/DOS user, I have to say that I do not feel that the 64K-segment, 640K-overall limitations of MS/DOS should in any way influence the design of a new typesetting system. Whilst I feel little affinity for the GUI-based nature of Microsoft Windows, its elimination of the 640K-limit for native-mode programs is such a step forward that I am prepared to argue that any future typesetting system for MS/DOS-based systems should assume the existence of Windows (or OS/2), or otherwise avoid the 640K barrier by using techniques such as that adopted by Eberhard Mattes' *emTEX386*.[12] If we continue to observe the constraints imposed by primitive systems such as MS/DOS, what hope have we of creating a typesetting system for the future rather than for yesterday?

These might be termed the historical (or 'necessary') deficiencies of TEX: deficiencies over which Knuth essentially had no control. But in examining the deficiencies of TEX, we must also look to the needs of its users, and determine where TEX falls short of these, regardless of the reasons. The term 'users', in this context, is all-encompassing, applying equally to the totally naïve user of LATEX and to the format designers themselves (people such as Leslie Lamport, Michael Spivak, and Frank Mittelbach); for although it is possible for format designers to conceal certain deficiencies in TEX itself (e.g. the lack of a `\loop` primitive), the more fundamental deficiencies will affect both. (Although it is fair to say that a sure sign of the skill of a format designer is the ease with which he or she can conceal as many of the apparent deficiencies as possible.) An excellent introduction to this subject is the article by Frank Mittelbach in *TUGboat*, 'E-TEX: Guidelines for future TEX' [2], and the subsequent article by Michael Vulis, 'Should TEX be extended?' [3]. Perhaps less accessible, and certainly more voluminous, are the combined submissions to NTS-L, which are archived at `TeX.Ac.Uk` as `Disk\$TeX:[TeX-Archive.Nts]Nts-L.All` and at `Ftp.Th-Darmstadt.De` as `/pub/tex/documentation/nts-l/*`.

So, what are these so-called 'fundamental deficiencies'? No doubt each of us will have his or her own ideas, and the three references cited above will serve as an excellent starting point for those who have never considered the subject before. What follows is essentially a very personal view — one person's ideas of what he regards as being truly fundamental. It is not intended to be exhaustive, nor necessarily original: some of the ideas discussed will be found in the references given; but I hope and believe that it is truly representative of current thinking on the subject. Without more ado, let us proceed to actual instances.

1. *The lack of condition/exception handling:* It is not possible within TEX to trap errors; if an error occurs, it invariably results in a standard error message being issued, and if the severity exceeds that of 'warning'[13] (e.g. overfull or underfull boxes), user interaction is required. This makes it impossible for a format designer to ensure that all errors are handled by the format, and actually prevents the adoption of adequate defensive programming techniques. For example, it is not possible for the designer of a font-handling system to trap an attempt to load a font which does not exist on the target system.
2. *The inability to determine that an error has occurred:* The `\last...` family (`\lastbox`, `\lastkern`, `\lastpenalty`, `\lastskip`) are unable to differentiate between the absence of a matching entity on the current list and the presence of a zero-valued entity; since there is all the difference in the world between a penalty

[11] Frank Mittelbach and Rainer Schöpf

[12] *emTEX386* uses a so-called 'DOS extender'.

[13] I use the VAX/VMS conventions of 'success', 'informational', 'warning', 'error' and 'severe error' as being reasonably intuitively meaningful here.

of zero and no penalty at all, vital information is lost.

3. *The hierarchical nature of line-breaking and page-breaking:* Once a paragraph has been broken into lines, it is virtually impossible to cause TEX to reconsider its decisions. Thus, when a paragraph spans two pages, the material at the top of the second page will have line breaks within it which are conditioned by the line breaks at the bottom of the previous page; this is indefensible, as the two occur in different visual contexts. Furthermore, it prevents top-of-page from being afforded special typographic treatment: for example, a figure may occur at the top of the second page, around which it is desired to flow text; if the paragraph has already been broken, no such flowing is possible (the issue of flowing text in general is discussed below). The asynchronous nature of page breaking also makes it almost impossible to make paragraph shape dependent on position: for example, a particular house style may require paragraphs which start at top of page to be unindented; this is non-trivial to achieve.
4. *The local nature of page breaking:* For anything which approximates to the format of a Western book, the verso-recto spread represents one obvious visual context. Thus one might wish to ensure, for example, that verso-recto pairs always have the same depth, even if that depth varies from spread to spread by a line or so. With TEX's present page breaking mechanism, allied to its treatment of insertions and marks, that requirement is quite difficult to achieve. Furthermore, by localising page breaking to the context of a single page, the risk of generating truly 'bad' pages is significantly increased, since there is no look-ahead in the algorithm which could allow the badness of subsequent pages to affect the page-breaking point on the current page.
5. *The analogue nature of 'glue':* TEX's fundamental paradigm, that of boxes and glue, provides an elegant, albeit simplistic, model of the printed page. Unfortunately, the flexible nature of glue, combined with the lack of any underlying grid specification, makes grid-oriented page layup impossible to achieve, at least in the general case. The present boxes and glue model could still be applicable in a grid-oriented version of TEX, but in addition there would need to be what might be termed 'baseline attractors': during the glue-setting phase, baselines would be drawn towards one of the two nearest attractors, which would still honour the constraints of `\lineskiplimit` (i.e. if the effect of drawing a baseline upwards were to bring two lines too close together, then the baseline would be drawn downwards instead).
6. *The lack of any generalised ability to flow text:* TEX provides only very simple paragraph shaping tools at the moment, of which the most powerful is `\parshape`; but one could envisage a `\pageshape` primitive and even a `\spreadshape` primitive, which would allow the page or spread to be defined as a series of discrete areas into which text would be allowed to flow. There would need to be defined a mechanism (not necessarily within the primitives of the language, but certainly within a kernel format) which would allow floating objects to interact with these primitives, thereby providing much needed functionality which is already present in other (mouse-oriented) systems.
7. *An over-simplistic model of lines of text:* Once TEX has broken paragraphs into lines, it encapsulates each line in an `\hbox` the dimensions of which represent the overall bounding box for the line; when (as is usually the case) two such lines occur one above the other, the minimum separation between them is specified by `\lineskiplimit`. If any two such lines contain an anomalously deep character on the first line, and/or an anomalously tall character on the second, then the probability is quite great that those two lines will be forced apart, to honour the constraints of `\lineskiplimit`; however, the probability of the anomalously deep character coinciding with an ascender in the line below, or of the anomalously tall character coinciding with a descender in the line above, is typically rather small: if TEX were to adopt a 'skyline'[14] model of each line, rather than the simplistic bounding-box model as at present, then such line pairs would not be forced apart unless it was absolutely necessary for legibility that they so be. Note that this does not require TEX to have any knowledge of the characters' *shape*; the present bounding-box model for characters is still satisfactory, at least for the purposes of the present discussion.
8. *Only partial orthogonality in the treatment of distinct entities:* TEX provides a reasonably orthogonal treatment for many of its entities

[14] This most apposite and descriptive term was coined by Michael Barr.

(for example, the \new... family of generators), but fails to extend this to cover all entities. Thus there is no mechanism for generating new instances of \marks, for example. Similarly, whilst \the can be used to determine the current value of many entities, \the \parshape returns only the number of ordered pairs, and not their values (there is no way, so far as can be ascertained, of determining the current value of \parshape). It is possible to \vsplit a \vbox (or \vtop), but not to *\hsplit an \hbox. The decomposition of arbitrary lists is impossible, as only a subset of the necessary \last... or \un... operators is provided. The operatorless implicit multiplication of <number><dimen-or-skip register> (yielding <dimen>) is also a source of much confusion; it might be beneficial if the concept were generalised to <number><register> (yielding <register-type>). However, this raises many related questions concerning the arithmetic capabilities of TEX which are probably superficial to our present discussion. I would summarise the main point by suggesting that orthogonality could be much improved.

9. *Inadequate parameterisation:* TEX provides a very comprehensive set of parameters with which the typesetting process may be controlled, yet it still does not go far enough. For example, one has \doublehyphendemerits which provide a numeric measure of the undesirability of consecutive hyphens; it might reasonably be posited that if two consecutive hyphens are bad, three are worse, yet TEX provides no way of indicating the increased undesirability of three or more consecutive hyphens. Also concerned with hyphenation is \brokenpenalty, which places a numeric value on the undesirability of breaking a page at a hyphen; again it might be posited that the undesirability of such a break is increased on a recto page (or reduced on a verso page), yet only one penalty is provided. A simple, but potentially infinite, solution would be to increase the number of parameters; a more flexible solution might be to incorporate the concept of formula-valued parameters, where, for example, one might write something analogous to \brokenpenalty = {\ifrecto 500 \else 200 \fi}, with the implication of delayed evaluation.
10. *Inadequate awareness of æsthetics:* TEX is capable of producing results which æsthetically are the equal or better of any computer typesetting system available today, yet the results may still be poorer than that achieved by more traditional means. The reason for this lies in the increased detachment of the human 'operator', who now merely conveys information to the computer and sits back to await the results. When typesetting was accomplished by a human compositor, he or she was aware not only of the overall shape of the text which was being created, but of every subtle nuance which was perceivable by looking at the shapes and patterns created on the page. Thus, for example, rivers (more or less obvious patterns of white space within areas of text, where no such patterns are intended), repetition (the same word or phrase appearing in visually adjacent locations, typically on the immediately preceding or following line), and other æsthetic considerations leapt out at the traditional typesetter, whereas TEX is blissfully unaware of their very existence. Fairly complex pattern matching and even image processing enhancements might need to be added to TEX before it was truly capable of setting work to the standards established by hot-metal compositors.

Clearly one could continue adding to this list almost indefinitely; every system, no matter how complex, is always capable of enhancement, and TEX is no exception to this rule. I have quite deliberately omitted any reference to areas such as rotated text and boxes, support for colour, or support for graphics, as I believe them to be inappropriate to the current discussion: they are truly *extensions* to TEX, rather than deficiencies which might beneficially be eliminated. But I believe I have established that there *are* areas in which TEX is capable of being improved, and would prefer to leave it at that.

This brings us therefore to the final theme: how should we proceed? The NTS-L approach is obviously helpful, in that it allows the entire (e-mail connected) TEX community to contribute to the discussion, but I see at least two problems:

1. Those who are not on e-mail[15] are essentially excluded from the discussion; I do not see any easy solution to this problem.
2. The views expressed are, in some cases, radically different, and I wonder whether we will ever converge on a universally acceptable decision.

The second is in many ways the more important issue (Knuth apart), for unless the decisions made are acceptable to a very large majority of the contributors, the group may split, with part electing to

[15] Knuth is not on e-mail...

go one route and another part electing to adopt a different strategy. This could result in a proliferation of *Über-TEXs*, with a concomitant fragmentation of the user community. Natural selection would surely winnow out the real non-starters before too long, but I seriously worry about the effect of such a proliferation on the TEX community, and even on TEX itself: after all, if we can't agree amongst ourselves whether there should be a successor to TEX, and if so what functionality it should possess, the whole credibility of the TEX ethos will be called into question. I would not like this to happen.

Somehow, therefore, we have to find a generally acceptable solution. My intuitive feeling is that such a solution will either be conservative or radical, but nothing in between. (This may seem like a distinct hedging of bets, but I hope that my meaning is clear: I believe that a compromise solution, which tries to be all things to all people, is doomed to failure.) I do truly believe that adopting both solutions (one conservative, one radical) may be the best way forward: as an initial step, we identify (as I have tried to do above) any true *deficiencies* of TEX — those that actually prevent it from accomplishing its stated aims — and rectify those, producing a system that is backwards compatible with present TEX implementations whilst being capable of achieving superior results. In parallel with this (which is intended to be a reasonably short term and straightforward project, requiring not too much in the way of resources), we start planning a truly radical New Typesetting System, with the same fundamental design desiderata as TEX (portability, freely available, fully documented, bug-free...), but designed for the technology of tomorrow[16] rather than that of today.

Considering first the conservative approach, we will need to identify what is feasible, as well as what is desirable. Clearly this will require advice from those who are truly familiar with `TEX.WEB`, as I see this approach purely as modifications to the WEB rather than as a re-write in any sense. Chris Thompson and Frank Mittelbach are obvious candidates here, and Frank is already a member of the NTS team; I would suggest that if we adopt this strategy, Chris be invited to participate as well. Once we have identified what is possible, we will need a reasonably accurate estimate of time-to-implement, and if this exceeds that which can be achieved with volunteer labour, we will need to seek funds to implement this solution. I would suggest that TUG be approached at this stage (obviously they will have been kept informed of the discussions), and asked if they are willing to fund the project. There seems no point in projecting beyond this stage in the present paper.

For the radical approach, familiarity with WEB is probably unnecessary, and indeed may be a disadvantage: if we are seeking a truly NEW Typesetting System, then detailed familiarity with current systems may tend to obfuscate the issue, and certainly may tend to constrain what should otherwise be free-ranging thoughts and ideas. We will need to consult with those outside the TEX world, and the advice of practising typographers[17] and (probably retired) compositors will almost certainly prove invaluable. But above all we will need people with vision, people who are unconstrained by the present limits of technology, and who are capable of letting their imagination and creativity run riot.

And what conclusions might such a group reach? Almost by definition, the prescience required to answer such rhetorical questions is denied to mere mortals; but I have my own vision of a typesetting system of the future, which I offer purely as an example of what a New Typesetting System might be. Firstly (and despite my quite ridiculous prejudices against windowing systems), I believe it will inherently require a multi-windowing environment, or will provide such an environment itself (that is, I require that it will make no assumptions about the underlying operating environment, but will instead make well-defined calls through a generic interface; if the host system supports a multi-windowing environment such as Microsoft Windows or the X Window System, the NTS will exploit this; if the host system does not provide such intrinsic support, then it will be the responsibility of the implementor to provide the multi-windowing facilities). I envisage that perhaps as many as eight concurrent displays might be required: linked graphic and textual I/O displays, through which the designer will be able to communicate the underlying graphic design in the medium of his or her choice (and observe in the other window the alternative representation of the design); an algorithmic (textual) display, through which the programmer will communicate how decisions are to be made; two source displays, one text, one graphic, through which the author will communicate the material to be typeset; and a preview display, through which an exact facsimile of the finished product may be observed at any desired level of detail. A further display will provide interaction (for example, the system might inform the user that some guidance is needed to place a particularly

[16] and beyond

[17] Michael Twyman and Paul Stiff have indicated a keen desire to be involved in the project.

tricky figure), and the last will enable the user to watch the system making decisions, without cluttering up the main interactive window. Needless to say, I assume that the system will essentially operate in real time, such that changes to any of the input windows will result in an immediate change in the corresponding output windows. I assume, too, that the input windows will be able to slave other unrelated programs, so that the user will be able to use the text and graphics editors of his or her choice. Of course, not all windows will necessarily be required by all users: those using pre-defined designs will not need either the design-I/O or the algorithm-input windows, and will be unlikely to need the trace-output window; but the interaction window may still be needed, and of course the source-input windows unless the source, too, has been acquired from elsewhere. For just such reasons, the system will be capable of exporting any designs or documents created on it in plain text format for import by other systems.

And underneath all this? Perhaps no more than a highly refined version of the TEX processor; totally re-written, probably as a procedural language rather than a macro language (why procedural rather than, say, list processing or declarative? to ensure the maximum acceptability of the system: there are *still* more people in the world who feel comfortable with procedural languages than with any of the other major genres), and obviously embodying at least the same set of enhancements as the interim conservative design, together with support for colour, rotation, etc. The whole system will, of course, be a further brilliant exposition of literate programming; will be placed in the public domain; will be capable of generating DVI files as well as enhanced-DVI and POSTSCRIPT; and will be so free of bugs that its creators will be able to offer a reward, increasing in geometric progression, for each new bug found...

But we will need one final element, and I have deliberately left this point to the very end: we will need the advice of Don Knuth himself. Don has now distanced himself from the TEX project, and is concentrating on *The Art of Computer Programming* once again. This detachment is very understandable — TEX has, after all, taken an enormous chunk out of his working (and, I suspect, private) life — and I hope that we all respect his wish to be allowed to return once again to 'mainstream' computer science, mathematics, and Bible study. But I think it inconceivable that we can afford to ignore his advice; and if I were to have one wish, it would be this: that I would be permitted to meet him, for whatever time he felt he could spare, and discuss with him the entire NTS project. I would like to know, above all, what changes *he* would make to TEX, were he to be designing it today, rather than fifteen years ago; I would like to know if he agrees that the deficiencies listed above (and those that appear elsewhere) are genuine deficiencies in TEX, or are (as I sometimes fear) simply the result of an inadequate understanding of the true power and capabilities of TEX; and I would like to know how he feels about the idea of an 'Extended TEX' and of a New Typesetting System (I suspect he would be far more enthusiastic about the latter than the former). And I suppose, if I am honest, I would just like to say 'Thank you, Don', for the countless hours, days, weeks, months and probably years of pleasure which TEX has given me.

Philip Taylor, July 1992

References

[1] Donald E. KNUTH: "The Future of TEX and METAFONT", in *TUGboat*, Vol. 11, No. 4, p. 489, November 1990.

[2] Frank MITTELBACH: "E-TEX: Guidelines for future TEX", in *TUGboat*, Vol. 11, No. 3, pp. 337–345, September 1990.

[3] Michael VULIS: "Should TEX be extended?", in *TUGboat*, Vol. 12, No. 3, pp. 442–447, September 1991.

[4] Zlatuška, Jiří (ed): *EuroTEX '92 Proceedings*, pp. 235–254, September 1992. Published by *CS*TUG, Czechoslovak TEX Users Group, ISBN 80-210-0480-0.

Appendix A

Inaugural members of the NTS-L team

- Rainer Schöpf (Chairman)
- Peter Abbott
- Peter Breitenlohner
- Frank Mittelbach
- Joachim Schrod
- Norbert Schwarz
- Philip Taylor

◇ Philip TAYLOR
Royal Holloway and Bedford New College,
University of London,
Egham Hill,
Egham,
Surrey, U.K.
<P.Taylor@Vax.Rhbnc.Ac.Uk>

Software

XBibTEX and Friends*

Nickolas J. Kelly and Christian H. Bischof

Abstract

The XBibTEX utility provides an X Window interface for inserting entries into a bibliography database in BIBTEX format. XBibTEX provides a template of the required and optional fields for each entry type defined by BIBTEX. In addition, XBibTEX accommodates two additional entries named "keyword" and "annote" for maintaining an annotated bibliography. The user enters the information to be stored in the database; XBibTEX automatically inserts the entry in the bibliography file in a format that conforms to BIBTEX's specifications. We also introduce two related utilities, **bibprocess** and **bibsearch**. **bibprocess** allows one to merge the contents of user-defined fields into a `.bbl` file produced by BIBTEX; **bibsearch** allows one to retrieve the LATEX key of publications that contain user-specified keywords. This article describes the features of these programs.

1 How to Use XBibTEX

While BIBTEX provides a convenient mechanism for storing and retrieving bibliographic records, it lacks a convenient means for entering new items. The user can easily make mistakes — for example, forgetting the required fields, the double quotes, or the commas surrounding the information in an entry.

XBibTEX minimizes these errors by

- displaying a template with all the fields defined for a particular type of bibliographic entry,
- making sure that information for required fields is indeed provided, and
- storing the information in a format that is guaranteed to be syntactically correct.

The 'look and feel' of XBibTEX is best conveyed through an example. After typing "xbibtex", the **Application Window** in Figure 1 will appear. This window contains a menu of options which the user may select by pressing the left mouse button over the appropriate option.

The *Article*, *Book*, *Inproceedings*, and *Techreport* buttons are bibliography entry types defined by BIBTEX. A menu containing the other (in our experience, less common) entry types known to XBibTEX pops up when the user presses the *Others* button.

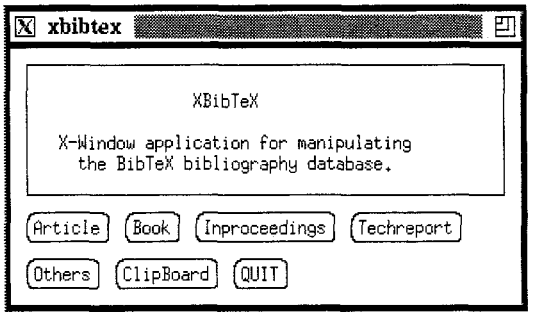


Figure 1: Application Window

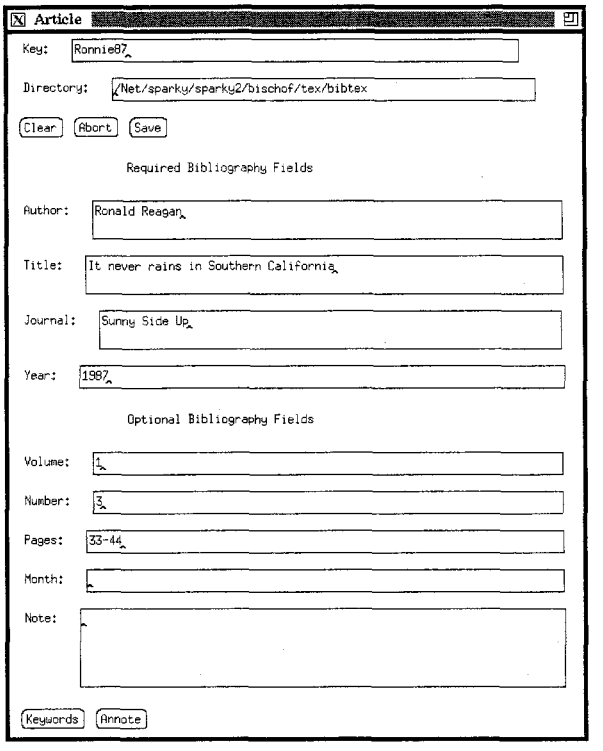


Figure 2: Entry Window for an *Article*

Let us assume we enter an article into our bibliography database. After pressing the *Article* button, the Article **Entry Window** displayed in Figure 2 will pop up, but with empty text fields. Notice that all of the buttons in the **Application Window** have been grayed except for the *ClipBoard* button. This "grey-out" prevents one from exiting the application while entering an entry.

There are two kinds of fields: "required" and "optional". The "required" fields must be filled in for XBibTEX not to complain. For example, let us assume that the author is Ronald Reagan; the title of the article is "It never rains in Southern California"; the date of publication is 1987; the page range is 33–44; and the volume, issue, and journal title

* This work was supported by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U. S. Department of Energy, under Contract W-31-109-Eng-38.

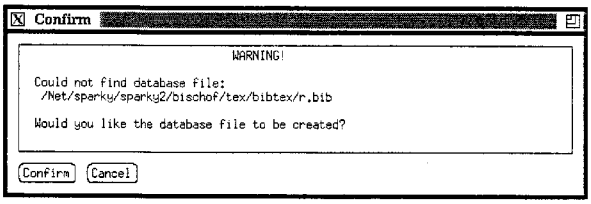


Figure 3: The Confirmation Window

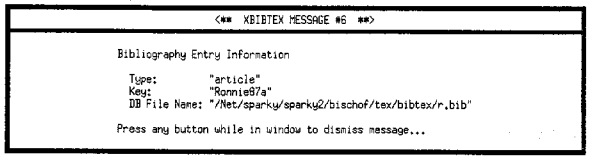


Figure 4: A Sample Message Window

are Vol. 1, No. 3, *Sunny Side Up*. Then one would enter the text as displayed in Figure 2. Note that there is no need to type the double quotes that surround information in BibTeX entries. XBibTeX will automatically generate them.

In addition, we have to specify a "key", and we enter "Ronnie87" (say). One has to specify a directory where the database (`.bib`) files are kept. The path can be an absolute path or can be relative to the directory from which XBibTeX was started. The path entered is not expanded by the shell, so the tilde (~) notation does not work, for example, to specify your home directory. Since database files are usually kept in one directory, the environment variable XBIBTEX_DATABASE_PATH can be set to specify this path so it need not be reentered every time.

When all the information has been entered, pressing the *Save* button causes the **Article Window** to disappear, and the **Confirmation Window** (see Figure 3) will appear since the database file `r.bib` does not yet exist. Upon confirmation, the **Message Window** (see Figure 4) will be displayed. The **Message Window** identifies the key that should be used from now on in a LaTeX `\cite` command to refer to this entry. Notice that the LaTeX key is not the same as the XBibTeX key that was entered — the LaTeX key is the XBibTeX key with an "*a*" appended to it. XBibTeX automatically indexes entries by appending a lower-case letter. This procedure allows for multiple entries to have the same XBibTeX key, but has the restriction that one can create only twenty-six entries with the same "base" XBibTeX key. The **Message Window** also shows in which file the new entry was stored. The database file name is determined by the first character of the XBibTeX key with the "`.bib`" extension appended.

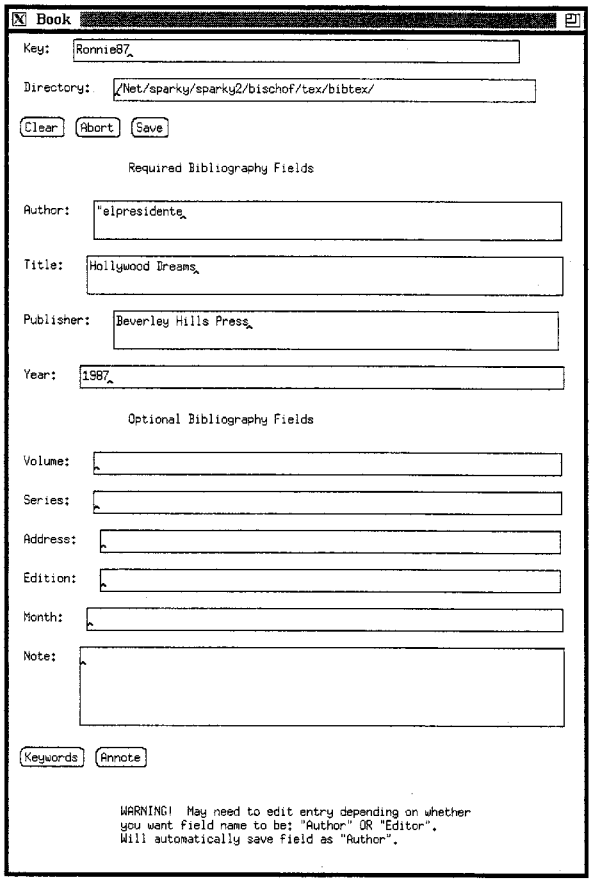


Figure 5: Entry Window for a *Book*

Now let us repeat the same procedure, except this time a book will be entered into the database. Let us assume the book is, say, "Hollywood Dreams", again by Ronald Reagan, published by Beverly Hills Press in 1987. After opening the "book" template by pressing the *Book* button in the **Applications Window**, we enter the information as shown in Figure 5. Note that we entered the same XBibTeX key as before, namely, "Ronnie87". Since there is already an entry that used the same XBibTeX key (and was stored with the LaTeX key "Ronnie87a"), the new entry will be assigned the LaTeX key "Ronnie87b". Also note that we entered `"elpresidente` in the author field. If an entry starts with a leading quote and has no spaces, newlines, or tabs in it, it will be saved without quotes surrounding it. Therefore, a declaration like

```
@string{elpresidente = "Ronald Reagan"}
```

causes this entry to be expanded correctly by BibTeX. The WARNING! message at the bottom of the template indicates that if one wanted to enter an 'editor' field instead of an 'author' field (BibTeX requires one of the two), one has to edit the `.bib` file by hand after storing this entry, since XBibTeX

assumes that the "author" field has been entered.

If we press the "annote" button, another window pops up into which additional information can be entered, which will be stored as an additional entry labeled "annote". BibTeX will ignore this entry since it is not one of those that BibTeX knows about. (The "bibprocess" utility described in Section 2 makes it possible to merge this information with output generated by standard BibTeX.)

After saving this entry, the contents of the BibTeX database file will look as follows, assuming that the file "r.bib" did not exist or was empty before, and "What great stuff!" was entered in the "annote" window.

```
@article{Ronnie87a,
 author = "Ronald Reagan",
 title = "It never rains in Southern California",
 journal = "Sunny Side Up",
 year = "1987",
 volume = "1",
 number = "3",
 pages = "33--44"
 }

@book{Ronnie87b,
 author = elpresidente,
 title = "Hollywood Dreams",
 publisher = "Beverly Hills Press",
 year = "1987",
 annote = "What great stuff!"
 }
```

This example provided a flavor of the functionality that XBibTeX affords. In summary, XBibTeX has the following features:

- Automatically provides a template of the required and optional fields associated with each entry type as defined by BibTeX.
- Ensures that the required fields of a specific entry are filled.
- Produces a unique LaTeX key by which entries are referenced.
- Automatically inserts entries in alphabetical order (by key) into the database.
- Provides a clipboard that can be initialized to display a specified file. This feature can be useful for displaying an abbreviations file from which to cut and paste.
- Provides two new fields, "annote" and "keywords", that are useful for maintaining an annotated bibliography.

2 bibprocess and bibsearch

In this section we describe two related utilities:

bibprocess: which allows one to merge the context of additional fields added to `.bib` files into the `.bbl` files produced by BibTeX, and

bibsearch: which allows one to search for the LaTeX keys of bibliography entries containing the desired information.

2.1 bibprocess

As mentioned above, XBibTeX allows one to enter two additional fields ("annote" and "keywords") which are ignored by BibTeX. **bibprocess** provides the functionality to merge the contents of those fields (as well as others) with the BibTeX output. Consider the output produced by BibTeX when we cite "Ronnie87a" and "Ronnie87b". (For future reference, assume that it was written to `test.bbl`.)

```
\begin{thebibliography}{1}

\bibitem{Ronnie87b}
Ronald Reagan.
\newblock {\em Hollywood Dreams}.
\newblock Beverly Hills Press, 1987.

\bibitem{Ronnie87a}
Ronald Reagan.
\newblock It never rains in Southern California.
\newblock {\em Sunny Side Up}, 1(3):33--44, 1987.

\end{thebibliography}
```

Since "annote" is a keyword that is not known to BibTeX, the information in the "annote" field (i.e., "What great stuff!") was ignored.

The **bibprocess** utility makes it possible to merge this information into the `.bbl` file. The syntax is

```
bibprocess
   <bibfile> [ <bibfile> ... ] <bblfile>
   -f <formatfile>
   -k <keyword> [ -k <keyword> ... ]
   [ -o <outputfile> ]
```

Given a `<bblfile>` (a file with a `.bbl` extension) produced by BibTeX from a set of `<bibfiles>` (files with a `.bib` extension), **bibprocess** appends the contents of the fields specified by `<keyword>` to the corresponding entries in `<bblfile>`. The output is written to `<outputfile>` if the `-o` option is present, and to `stdout` otherwise. The `<formatfile>` specifies how the new information is to be formatted.

Again, it's easiest to understand this process with an example. Assume that we wish to merge the "keywords" and "annote" information that XBibTeX allowed us to specify. The original BibTeX output is in `test.bbl`, the bibliography

file is `r.bib`, and the merged output should go to `testmerged.bbl` (say). We issue the command

```
bibprocess r.bib test.bbl -f merge.fmt
       -k keywords -k annote
           -o testmerged.bbl
```

The file `merge.fmt` contains the following information:

```
annote
before: @\newblock {{\bf annote:} @
after: @}@

keywords
before: @\newblock {{\bf keywords:} @
after: @}@
```

We see that we have an entry for each keyword that is to be merged, and for each keyword we specify what string to place *before* and *after* the information to be merged. The "at" sign `@` is used as a delimiter, the entries must be separated by blank lines, and the `<keywords>` ("annote" and "keywords" in our example) must be specified with lower-case characters. `testmerged.bbl` then contains the following information:

```
\begin{thebibliography}{1}

\bibitem{Ronnie87b}
Ronald Reagan.
\newblock {\em Hollywood Dreams}.
\newblock Beverly Hills Press, 1987.
\newblock {{\bf annote:} What great stuff!}

\bibitem{Ronnie87a}
Ronald Reagan.
\newblock It never rains in Southern California.
\newblock {\em Sunny Side Up}, 1(3):33--44, 1987.

\end{thebibliography}
```

We see that the "annote" information has been appended to the "Ronnie87a" entry using the format specified in `merge.fmt`. There was no "keywords" information, so no field was created for it.

Using **bibprocess** you can thus augment the functionality of BibTeX and augment its output with any number of user-defined fields.

2.2 bibsearch

The syntax of **bibsearch** is as follows:

```
bibsearch <bibfile> -k <keyword>
          [-k <keyword>] [-o <outputfile>]
          -s <searchword>
```

bibsearch searches the `title` field and the fields specified by `<keyword>` in a bibliography file `<bibfile>` for an occurrence of `<searchword>` and prints the LaTeX keys of all entries either to `<outputfile>` or `stdout`. For example,

```
bibsearch r.bib -k annote -s "great stuff"
```

will produce

```
Ronnie87b
```

on the standard output. Note that one must enclose the search string in quotes if it contains more than one word.

In summary, **bibsearch** is useful for retrieving citations containing information on specified words — in particular, if one maintains a "keywords" entry in the BibTeX database.

3 How to Obtain XBibTeX and Friends

The source for XBibTeX, **bibprocess**, and **bibsearch** as well as a more detailed user and installation guide can be retrieved via anonymous ftp from `info.mcs.anl.gov` (current Internet address: 140.221.10.1). The `pub/xbibtex` subdirectory contains the following files:

userguide.ps.Z: A more complete description of XBibTeX, **bibprocess**, and **bibsearch**, as well as installation instructions.

xbibtex.shar.Z: Source code, man page, and makefile for XBibTeX.

bibutils.shar.Z: Source code, man pages and makefiles for **bibprocess** and **bibsearch**.

All files must be transferred in 'binary' mode.

◇ Nickolas J. Kelly
Nielsen Advanced Information Technology Center
Bannockburn Lake Office Plaza
2345 Waukegan Rd.
Bannockburn, IL 60015

◇ Christian H. Bischof
Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, Illinois 60439-4801
bischof@mcs.anl.gov

Searching in a DVI File

Nigel Chapman

1. Introduction

Most, if not all, DVI previewers and printer drivers provide a facility for selecting a subset of the pages of a document; this subset is specified using the contents of the `\count0` to `\count9` registers that TEX outputs to identify each page of the file. This makes it easy to preview just pages 7, 8 and 9, but what if you know you want to look at the page with the paragraph about Katzenellenbogen by the Sea? If you're not sure how the page makeup worked out, you won't know where that is. Trial and error will find the right page sooner or later, but it would be more convenient if there was a facility for selecting a page by its content, that is, the occurrence on it of a particular string.

Many efficient string searching algorithms already exist; they are used routinely in text editors and other programs. These algorithms take as their input a string of characters — the target — and a pattern. The pattern specifies a set of strings. The task of the searching algorithm is to find the location, if any, within the target of a substring that belongs to the set specified by the pattern. The pattern may simply be a single string, specifying itself, or it may use metacharacters and some formalism such as regular expressions to specify a larger set of strings. In general, the more elaborate the language permitted for specifying patterns, the more elaborate the search algorithm will be. There are well known efficient algorithms for searching for single strings [4] and for sets specified by regular expressions [1].

A DVI file is a sequence of typesetting commands, some of which may have parameters. (DVI commands are fully described in [3,§§583–590].) A user specifying a pattern to search for will want to type that pattern at a terminal using the subset of ASCII that corresponds to printable characters. Thus, before one of the standard string searching algorithms can be employed, either the pattern must be converted to a sequence of DVI commands, the DVI file must be mapped into a string of printable ASCII characters, or both DVI file and pattern must be mapped into some other common representation.

Leaving aside the possibility of using anything more elaborate than simple strings as patterns, a possible approach based on the first of these options is to use TEX to convert the pattern into DVI. Using this approach, it would be possible for patterns to be specified in the TEX language, and thus to make use of macros and to carry out searches on all features of a document, including math mode material and even rules and spaces. Search patterns could be extracted directly from the TEX source of a document. (In fact, for non-trivial search strings they would probably have to be, because of the difficulty of deciding exactly what TEX commands produced some particular output.) However, the problems of interfacing TEX are considerable, and the overheads of running it to process every search string are unlikely to be acceptable. Furthermore, the actual DVI produced by TEX for a particular string will depend on the context in which that string appears. Such elements as interword spacing, line breaking and hyphenation may be very different when the string appears in the middle of a paragraph and when it is typeset in isolation. Thus, even after a pattern was converted to DVI, it would not be possible to apply simple string matching: some sort of fuzzy matching would be necessary.

If converting the pattern to DVI is problematical, what about converting the DVI file to ASCII? This is essentially the same task as that performed by DVI previewers for dumb terminals, and it suffers from the same limitations: only text material can be dealt with properly, and spacing must be approximated. It has the great virtue of being simple, and, once the transformation has been done, any string matching algorithm can be used, including those that support regular expressions. This approach will be looked at further in the next section.

Finally, there is the possibility of converting both the DVI file and the pattern to some common representation. The obvious choice here is the extended character code set used by TEX to specify characters in its math symbol and extension fonts. This requires some means of specifying characters in the pattern other than printable ASCII characters; the obvious way of doing this is by permitting a suitable subset of TEX commands to be used in patterns. Matching can then be done on text and math mode material. This approach is further described in section 3.

2. Text Searching

The text parts of a TEX document will be typeset using TEX's text fonts. Computer Modern text fonts each contain 128 characters and these are laid out in such a way that the printable ASCII characters occur in the positions corresponding to their ASCII codes. The positions corresponding to the ASCII non-printable (control) characters are used for ligatures, accents and Greek letters (see the

font layout tables in Appendix F of [2]). A search pattern typed by a user will only use printable ASCII characters, but it is necessary to ensure that, for example, the pattern `ffi` matches the ligature ffi.

A DVI file can be thought of as a program in the machine code of a virtual typesetting machine. The instructions of this machine perform primitive typesetting operations, such as *set a character* or *select a new font*. These instructions are held in the DVI file as bit patterns. For reading by people, they can be represented by mnemonics, just as conventional machine code instructions are given mnemonic form in assembly language. DVI instructions may have parameters, specifying, for example, the length of a rule to be set, or the code for a character. There are two sets of DVI commands that cause a character to be typeset: the *set* commands, with mnemonics `set_char_0` through `set_char_127`, `set1`, `set2`, `set3` and `set4`, and the *put* commands, `put1`, `put2`, `put3` and `put4`. The first 128 of these are unparameterized and directly specify a character code, the others take an argument of between one and four bytes. TEX always uses `set_char_`n for a character with code n, if $0 \leq n \leq 127$, so any ASCII character will be set with either a `set_char_`n or `put1` (see [3,§585]). Because of kerning, commands for spacing may appear between character setting commands, even within a word. For example, the DVI commands corresponding to the word 'Katzenellenbogen', as it appears near the beginning of the file containing this paper, have the following mnemonic representation.

```
setchar75
setchar97
setchar116
setchar122
setchar101
setchar110
setchar101
setchar108
setchar108
setchar101
setchar110
right2 -18205
setchar98
x0 18205
setchar111
setchar103
setchar101
setchar110
```

(The `right2` and `x0` commands perform horizontal spacing.)

A possible strategy for converting a DVI file into a stream of characters is to scan the file, ignoring everything except *set* and *put* commands, and return the corresponding ASCII code. But this doesn't do quite what is required. In the first place, all characters not in text fonts should also be skipped. To do this, it is necessary to keep track of the current font by interpreting the additional DVI commands `fnt_def`, `fnt` and `fnt_num`. The external name of each font used can be found among the parameters of its `fnt_def`; it is necessary to have *a priori* knowledge of which of these names correspond to text fonts — for Computer Modern fonts this knowledge is obtainable in Appendix F of *The TEXbook* [2].

Secondly, ligatures should be expanded into their component letters, so, for example, when a `set_char_14` command is found, it should be converted to the three letters `f f i`. Dashes should also be expanded, en-dash to `--` and em-dash to `---`. Dotless i and j should be replaced by ordinary i and j. All other non-printable characters will have to be skipped, as should the Spanish punctuation marks ¡ and ¿ and hyphens, the last since a pattern should match irrespective of whether the corresponding occurrence in the DVI file is split across lines. Strictly speaking, to skip hyphens it is necessary to know the `\hyphenchar` of the current font, and this can only be determined by examining the format used in TEXing the document. It will usually be safe, though, to assume it is `'\-`.

If the scheme described is used, a stream of characters can be produced from the DVI file, corresponding to the textual parts of the document, and these can be matched against a search string made up of printable ASCII characters. However, spacing in the DVI file will be entirely skipped, so ASCII space characters in the search string will not match anything. One response to this is to insert space characters in places that correspond to word breaks in the DVI file. Because of what TEX does with glue, however, these cannot be identified with certainty (as the spacing produced by ASCII DVI previewers such as `dvitty` testifies). The easier alternative is to remove all spaces from the search pattern. This will produce spurious matches, e.g., `pullover` will match `pull over`, and vice versa, but this is preferable to failing to match because of incorrectly inserted spaces.

By allowing an escape character in the search pattern, this scheme can be extended to cope with the full range of accents in the Computer Modern text fonts. The obvious scheme is to use plain

TEX's control sequences for accents and accented characters and expand them in the pattern.

Use of negative glue, `\llap`, characters with negative width, and so on can lead to the commands for setting characters occurring in the DVI file in a very different order from that in which the characters appear in the typeset document. Thus, the matching obtained can only ever be approximate unless DVI commands are first sorted by x and y coordinates. For most applications, this probably wouldn't matter, since most searches will be for obviously identifiable words.

3. Searching in Mathematics

A DVI searching program based on the previous section will suffice to find the page with the text 'Katzenellenbogen by the Sea', but what about the page with

$$\left(\sum_{i=1}^{n} \alpha_i^{m_1} + \beta_i^{m_2}\right)^k ?$$

This is much more of a challenge. To begin with: how should the search pattern be specified? The obvious way is to use the TEX language, but this is not, in fact, an answer, because of TEX's powerful macro facilities and its almost unbounded possibilities for redefining the meaning of any character. One must assume at least part of a format in interpreting a search pattern. The obvious choice is the plain format, so that a search string for the above could be `$(\sum_{i=1}^n \alpha_i^{m_1} + \beta_i^{m_2})^k$`. This illustrates several of the problems that must be overcome when searching in math mode material.

Firstly, `\alpha` and `\beta` represent characters from TEX's math italic font; these occur in the same place as the ff and ffi ligatures in a text font and the two have to be distinguished. TEX does this by using mathcodes, which specify a font family and the position in the font (they also specify a type, but that is irrelevant here) and defining control sequences such as `\alpha` using `\mathchardef`. The mathcode values for such symbols can be used by the searching algorithm, provided the DVI file can be turned into a stream of mathcodes, rather than a stream of ASCII characters as described in the previous section. This is not much more difficult than determining when a character occurs in a text font. By keeping track of the external name of the current font it is possible to deduce the current font family; to do this properly, it is necessary to look at the format used to typeset the document and interpret any assignments to `\textfont`k. The lazy alternative is to assume the assignments of plain TEX. As all text fonts are assigned to family 0, the text matching using ASCII codes will still work.

Not all the special symbols available in math mode are defined by `\mathchardef`. Some are built up as a combination of other symbols, for example $\longmapsto$. Such composite symbols must be expanded into their components.† Some of these combinations are defined in quite an elaborate way, but it is relatively easy to deduce the sequence of character setting DVI instructions they will produce.

Multiple sub- and superscripts present a different sort of problem. A term such as x_i^2 can be produced in TEX by either `$x_i^2$` or `$x^2_i$`. Either pattern should match the DVI for x_i^2, irrespective of how that was specified in the original TEX source. Study of Appendix G of *The TEXbook* reveals that when an atom has both a subscript and a superscript, TEX always sets the superscript first, followed by the subscript. Thus, a pattern with both can be normalized to a form with the superscript first. The `_` and `^` can be ignored and the sequence of component characters, in the appropriate families, can be searched for. Again, this may produce spurious matches (`$x_i^2$` will match $x2i$, for example) but will not fail to match when it should.

The last major complication of searching in math mode is caused by those characters that can change their size. (How big are the parentheses in the example at the beginning of this section?) These include the things defined by `\delimiter`, `\mathaccent` and `\radical`, which may give two different mathcode values for the same control sequence. Matters are made more complicated by the fact that any character may be the first in a linked sequence of characters or may be constructed out of several pieces. These series and extensible characters are considered part of the font, and information about them has to be taken from its TFM file.

A control sequence defined by `\delimiter`, or a mathcode that specifies a character that is part of a sequence can be thought of as a pattern that describes a set of alternatives. An extensible character also specifies a set, but, in theory at least, it is an infinite set of strings of the form TR^kMR^kB or TR^kB where T, R, M and B are the top, repeating, middle and bottom pieces of the extensible character, and $k \geq 0$. All these possibilities can be described by regular expressions.

† A seemingly intractable problem here is the underline symbol, represented by `\_` in plain TEX, which is actually a rule.

$$
\begin{aligned}
\mathit{pattern} &\rightarrow \mathit{pterm}\ \{\,\underline{|}\ \mathit{pterm}\}\\
\mathit{pterm} &\rightarrow \mathit{pfactor}\ \{\,\mathit{pfactor}\}\\
\mathit{pfactor} &\rightarrow \mathit{pprimary}\ [\,\underline{*} \mid \underline{?}\,]\\
\mathit{pprimary} &\rightarrow \underline{\{}\,[\,\mathit{pattern}\,]\,\underline{\}} \mid \underline{\$}\ \mathit{mathpattern}\ \underline{\$}\\
&\quad \mid \mathit{pelement}\\
\mathit{pelement} &\rightarrow \underline{\mathrm{char}} \mid \underline{\mathrm{cs}}\\
\mathit{mathpattern} &\rightarrow \mathit{mathelement}\ \{\,\mathit{mathelement}\}\\
\mathit{mathelement} &\rightarrow \mathit{pelement}\,[\,\mathit{scription}\,]\\
&\quad \mid \underline{\{}\ \mathit{mathpat}\,[\,\mathit{scription}\,]\,\underline{\}}\\
\mathit{scription} &\rightarrow \underline{\uparrow}\ \mathit{scriptelement}\,[\,\underline{=}\mathit{scriptelement}\,]\\
&\quad \mid \underline{=}\ \mathit{scriptelement}\,[\,\underline{\uparrow}\mathit{scriptelement}\,]\\
\mathit{scriptelement} &\rightarrow \mathit{pelement} \mid \underline{\{}\,[\,\mathit{mathpat}\,]\,\underline{\}}\\
\mathit{mathpat} &\rightarrow \mathit{mathelement}\ \{\,\mathit{mathelement}\}
\end{aligned}
$$

Figure 1. Grammar for search patterns.

A delimiter specifies $d_1 \mid d_2$ where d_1 and d_2 are the small and large versions of the symbol; a series of characters is just $c_1 \mid c_2 \mid \ldots \mid c_n$, and an extensible recipe is $(TR^*MR^*B) \mid (TR^*B)$, which may be simplified to $TR^*(M \mid \epsilon)R^*B$ (ϵ is the empty string). Thus if patterns are specified as regular expressions, and searching is done using finite state machines, control sequences and characters corresponding to symbols that can change size one way or another can be treated as shorthands for these regular expressions. Note that the fact that the regular expression for an extensible character does not enforce the restriction that the number of repeatable segments above and below the middle must be the same doesn't matter, because no other combination can occur.

In summary, to deal with math mode, the DVI file must be converted to mathcodes specifying font family and character position and the pattern must be modified so that characters and control sequences in it are mapped into regular expressions matching such codes, taking into account the font family characters will be typeset in, delimiter codes, character series and extensible characters.

4. An Implementation

A prototype DVI searching facility based on the ideas in the previous sections has been implemented as part of a TEX-based hypertext system I am developing. Search patterns may be written in a language defined by the extended BNF grammar in Figure 1. (The symbols [,], {, } and | are grammar metacharacters: items enclosed in [and] are optional, those in { and } may occur zero or more times, | separates alternatives. The terminal symbols — the symbols of the pattern language — are underlined. Note the difference between the metacharacters {, } and | and the terminals $\underline{\{}$, $\underline{\}}$ and $\underline{|}$.) In patterns, the $\underline{|}$ operator separates alternatives, the postfix $\underline{*}$ means that the preceding pattern element may occur zero or more times, and the $\underline{?}$ that it is optional. Curly brackets are used to group items: the syntax of the pattern language means that they work as expected to delimit complex sub- and superscripts, but they are also used as brackets to override the default associativity of regular expression operators, in the usual way. The terminals $\underline{\mathrm{char}}$ and $\underline{\mathrm{cs}}$ represent lexical classes consisting of single characters other than pattern metacharacters and TEX-style control sequences, respectively. The metacharacters are represented in patterns by \\, *, \?, etc.

A pattern input by the user is parsed by a simple recursive descent parser, and a data structure for a nondeterministic finite state machine with ϵ transitions is constructed in a syntax-directed manner using the conventional construction (see [1]). When the lowest level parsing function recognizes a control sequence or an extensible character, it returns a primitive finite state machine to recognize the strings described by it. For most control sequences defined by mathchardefs this is just a two state, one transition machine that recognizes the corresponding mathcode. For delimiters and other characters that change their size in different ways, and for composite characters, a more elaborate machine, corresponding to the appropriate regular expressions, as described in section 3, are produced. For any single, non-extensible character, a machine to recognize the character in the appropriate family is constructed. A special control sequence `\any` is recognized; it matches any single symbol.

The code to construct the finite state machines for control sequences and so on is derived from the definitions in `plain.tex` and PL files for the Computer Modern fonts. If a more general facility were desired, tables would have to be constructed automatically from any user-specified TEX format and TFM or PL files, a task requiring non-trivial analysis of these files.

Once the nondeterministic finite state machine with ϵ-transitions has been constructed, the ϵ-transitions are removed, but the machine is not made deterministic. The nondeterministic machine is used directly in the searching process, by keeping track of a set of active states. State transitions are performed by calling a function that returns the next symbol from the DVI file. This function performs the mapping to a sequence of mathcode values as described previously. It is assumed that only Computer Modern fonts are used, and the font

family assignments of plain TEX are wired into the code.

A problem not previously mentioned is that of displaying the location of the matching string if one is found. This facility was felt to be useful. In order to do it, it is necessary to interpret enough of the DVI commands to keep track of the x and y coordinates at which each character would be displayed. When the finite state machine accepts a string, the coordinates of its end are known and an arrow can be displayed pointing back at the matched string. A more elegant method, such as highlighting the entire match in reverse video, requires finding the coordinates of the start of the string too, which is more difficult.

The majority of the code implementing DVI searching is concerned with parsing patterns, constructing nondeterministic finite state machines and removing ϵ-transitions — that is, the code that any searching program based on finite state machines would require. Most of the overhead specifically resulting from the application to DVI searching is in the code to produce primitive machines for control sequences and extensible characters. The code required to transform the DVI into a stream of mathcodes is relatively simple.

5. Conclusion

The implementation described demonstrates that it is possible to search in a DVI file, using the ideas presented here. Consideration of this implementation suggests that text searching is a feature well worth implementing, but that searching in math mode is less clearly worthwhile.

As should be evident, searching in mathematics adds a good deal of complexity and requires large amounts of code for dealing with mathematical control sequences. This in turn requires the manual or automatic processing of TEX formats and PL or TFM files. Extensible characters pretty much dictate the use of finite state machines for searching, rather than some simpler algorithm such as Knuth-Morris-Pratt. Searching in maths also introduces an undesirable feature into the searching interface: some mathematical control sequences are recognized, but others are not. It might seem capricious to a user that it is all right to use `\alpha` in a search pattern, but not, for example, `\pmatrix`. Furthermore, if you actually did want to search for the page containing

$$\begin{pmatrix} x-\lambda & 1 & 0 \\ 0 & x-\lambda & 1 \\ 0 & 0 & x-\lambda \end{pmatrix}$$

it would be no trivial task to construct a suitable pattern out of the facilities available.

However, searching in text is quite another matter. Implementing the transformation of the textual parts of the DVI file to ASCII is very simple. This can easily be combined with any string searching algorithm that scans from left to right to produce an efficient text searching facility. In practice, this is likely to be adequate to select any page of a document by content. As well as the experimental system described in section 4, the `dviscr` previewer distributed with `emtex` already supports basic text searching, correctly handling accents and ligatures. It is to be hoped that, in future, text searching will become a common enhancement to DVI previewers and printer drivers.

References

1. Alfred V. Aho, John E. Hopcroft and Jeffrey D. Ullman, *The Design and Analysis of Computer Algorithms.* Reading, Mass.: Addison-Wesley, 1974.
2. Donald E. Knuth, *Computers and Typesetting, Volume A, The TEXbook.* Reading, Mass.: Addison-Wesley, 1986.
3. Donald E. Knuth, *Computers and Typesetting, Volume B, TEX: The Program.* Reading, Mass.: Addison-Wesley, 1986.
4. Klaus Pirklbauer, 'A study of pattern-matching algorithms', *Structured Programming* **13**(2), 89–98, 1992.

◇ Nigel Chapman
Department of Computer Science,
University College London,
Gower Street,
London, WC1E 6BT
U.K.
Janet: `N.Chapman@uk.ac.ucl.cs`

Hyphenation Exception Log

Barbara Beeton

This is the periodic update of the list of words that TEX fails to hyphenate properly. The list last appeared in *TUGboat* 10, no. 3, starting on page 336. Everything listed there is repeated here. Owing to the length of the list, it has been subdivided into two parts: English words, and names and non-English words that occur in English texts.

This list is specific to the hyphenation patterns that appear in the original `hyphen.tex`, that is, the patterns for U.S. English. If such information for other patterns becomes available, consideration will be given to listing that too. (See below, "Hyphenation for languages other than English".)

In the list below, the first column gives results from TEX's `\showhyphens{...}`; entries in the second column are suitable for inclusion in a `\hyphenation{...}` list.

In most instances, inflected forms are not shown for nouns and verbs; note that all forms must be specified in a `\hyphenation{...}` list if they occur in your document.

Thanks to all who have submitted entries to the list. Since some suggestions demonstrated a lack of familiarity with the rules of the hyphenation algorithm, here is a short reminder of the relevant idiosyncrasies. Hyphens will not be inserted before the number of letters specified by `\lefthyphenmin`, nor after the number of letters specified by `\righthyphenmin`. For U.S. English, `\lefthyphenmin=2` and `\righthyphenmin=3`; thus no word shorter than five letters will be hyphenated. (For the details, see *The TEXbook*, page 454. For a digression on other views of hyphenation rules, see below under "English hyphenation".) This particular rule is violated in some of the words listed; however, if a word is hyphenated correctly by TEX except for "missing" hyphens at the beginning or end, it has not been included here.

Some other permissible hyphens have been omitted for reasons of style or clarity. While this is at least partly a matter of personal taste, an author should think of the reader when deciding whether or not to permit just one more break-point in some obscure or confusing word. There really are times when a bit of rewriting is preferable.

One other warning: Some words can be more than one part of speech, depending on context, and have different hyphenations; for example, 'analyses' can be either a verb or a plural noun. If such a word appears in this list, hyphens are shown only for the portions of the word that would be hyphenated the same regardless of usage. These words are marked with a '*'; additional hyphenation points, if needed in your document, should be inserted with discretionary hyphens.

The reference used to check these hyphenations is *Webster's Third New International Dictionary*, Unabridged.

English hyphenation

It has been pointed out to me that the hyphenation rules of British English are based on the etymology of the words being hyphenated as opposed to the "syllabic" principles used in the U.S. Furthermore, in the U.K., it is considered bad style to hyphenate a word after only two letters. In order to make TEX defer hyphenation until after three initial letters, set `\lefthyphenmin=3`.

Of course, British hyphenation patterns should be used as well. A set of patterns for UK English has been created by Dominik Wujastyk and Graham Toal, using Frank Liang's `PATGEN` and based on a file of 114925 British-hyphenated words generously made available to Dominik by Oxford University Press. (This list of words and the hyphenation break points in the words are copyright to the OUP and may not be redistributed.) The file of hyphenation patterns may be freely distributed; it is stored in the UK TEX Archive at Aston and may be retrieved by anonymous FTP.

Hyphenation for languages other than English

Patterns now exist for many languages other than English. The Stuttgart archive

`ftp.uni-stuttgart.de`

holds an extensive collection in the directory

`soft/tex/languages/hyphenation`

The List — English words

`academy(ies)`	acad-e-my(ies)
`addable`	add-a-ble
`ad-di-ble`	add-i-ble
`adrenaline`	adren-a-line
`af-terthought`	af-ter-thought
`agronomist`	agron-o-mist
`am-phetamine`	am-phet-a-mine
`anal-yse`	an-a-lyse
`anal-y-ses`	analy-ses *
`anomaly(ies)`	anom-aly(ies)
`an-tideriva-tive`	an-ti-deriv-a-tive
`anti-nomy(ies)`	an-tin-o-my(ies)

antin-u-clear	an-ti-nu-clear
antin-u-cleon	an-ti-nu-cle-on
an-tirev-o-lu-tion-ary	an-ti-rev-o-lu-tion-ary
apotheoses	apoth-e-o-ses
apotheo-sis	apoth-e-o-sis
ap-pendix	ap-pen-dix
archipelago	arch-i-pel-ago
archety-pal	ar-che-typ-al
archetyp-i-cal	ar-che-typ-i-cal
arc-t-an-gent	arc-tan-gent (better: arc tangent)
assignable	as-sign-a-ble
as-sig-nor	as-sign-or
as-sis-tantship	as-sist-ant-ship
asymp-tomatic	asymp-to-matic
asymp-totic	as-ymp-tot-ic
asyn-chronous	asyn-chro-nous
atheroscle-ro-sis	ath-er-o-scle-ro-sis
at-mo-sphere	at-mos-phere
at-tributed	at-trib-uted
at-tributable	at-trib-ut-able
avoirdupois	av-oir-du-pois
awo-ken	awok-en
ban-dleader	band-leader
bankrupt(cy)	bank-rupt(-cy)
ba-ronies	bar-onies
base-li-neskip	\base-line-skip
bathymetry	ba-thym-e-try
bathyscaphe	bathy-scaphe
bea-nies	bean-ies
be-haviour	be-hav-iour
be-vies	bevies
bib-li-ographis-che	bib-li-o-gra-phi-sche
bid-if-fer-en-tial	bi-dif-fer-en-tial
bil-l-able	bill-able
biomath-e-mat-ics	bio-math-e-mat-ics
biomedicine	bio-med-i-cine
biorhythms	bio-rhythms
bitmap	bit-map
blan-der	bland-er
blan-d-est	bland-est
blin-der	blind-er
blon-des	blondes
blueprint	blue-print
bornolog-i-cal	bor-no-log-i-cal
bo-tulism	bot-u-lism
brus-quer	brusquer
bus-ier	busier
bus-i-est	busiest
buss-ing	bussing
but-ted	butted
buz-zword	buzz-word
ca-caphony(ies)	ca-caph-o-ny(ies)
cam-er-a-men	cam-era-men
cartwheel	cart-wheel
catar-rhs	ca-tarrhs
catas-trophic	cat-a-stroph-ic
catas-troph-i-cally	cat-a-stroph-i-cally
cauliflower	cau-li-flow-er
cha-parral	chap-ar-ral
chartreuse	char-treuse
cholesteric	cho-les-teric
cigarette	cig-a-rette
cin-que-foil	cinque-foil
cognac	co-gnac
cog-nacs	co-gnacs
com-parand	com-par-and
com-para-nds	com-par-ands
comptroller	comp-trol-ler
con-formable	con-form-able
con-formist	con-form-ist
con-for-mity	con-form-ity
congress	con-gress
cose-cant	co-se-cant
cotan-gent	co-tan-gent
crankshaft	crank-shaft
crocodile	croc-o-dile
crosshatch(ed)	cross-hatch(ed)
dachshund	dachs-hund
database	data-base
dat-a-p-ath	data-path
declarable	de-clar-able
defini-tive	de-fin-i-tive
delectable	de-lec-ta-ble
democratism	de-moc-ra-tism
de-mos	demos
deriva-tive	de-riv-a-tive
diffract	dif-fract
di-rer	direr
di-re-ness	dire-ness
dis-parand	dis-par-and
dis-traugh-tly	dis-traught-ly
dis-tribute(d)	dis-trib-ute(d)
dou-blespace(ing)	dou-ble-space(-ing)
dol-lish	doll-ish
drif-tage	drift-age
driver(s)	dri-ver(s)
dromedary(ies)	drom-e-dary(ies)
duopolist	du-op-o-list
duopoly	du-op-oly
eco-nomics	eco-nom-ics
economist	econ-o-mist
elec-trome-chan-i-cal	electro-mechan-i-cal
elec-tromechanoa-cous-tic	electro-mechano-acoustic
eli-tist	elit-ist
en-trepreneur(ial)	en-tre-pre-neur(-ial)
epinephrine	ep-i-neph-rine
equiv-ari-ant	equi-vari-ant
ethy-lene	eth-yl-ene
ev-ersible	ever-si-ble
ev-ert(s,ed,ing)	evert(s,-ed,-ing)
exquisite	ex-quis-ite
ex-traor-di-nary	ex-tra-or-di-nary
fermions	fermi-ons
flag-el-lum(la)	fla-gel-lum(-la)
flammables	flam-ma-bles
fledgling	fledg-ling
flowchart	flow-chart
formidable(y)	for-mi-da-ble(y)
forsythia	for-syth-ia
forthright	forth-right
freeloader	free-loader
friendlier	friend-lier
frivolity	fri-vol-ity
frivolous	friv-o-lous
ga-some-ter	gas-om-e-ter
geodesic	ge-o-des-ic
geode-tic	ge-o-det-ic

ge-o-met-ric	geo-met-ric
geotropism	ge-ot-ro-pism
gnomon	gno-mon
grievance	griev-ance
grievous(ly)	griev-ous(-ly)
hairstyle	hair-style
hairstylist	hair-styl-ist
harbinger	har-bin-ger
harlequin	har-le-quin
hatcheries	hatch-eries
hemoglobin	he-mo-glo-bin
hemophilia	he-mo-phil-ia
hep-atic	he-pat-ic
hermaphrodite(ic)	her-maph-ro-dite(-ic)
heroes	he-roes
hex-adec-i-mal	hexa-dec-i-mal
holon-omy	ho-lo-no-my
ho-mo-th-etic	ho-mo-thetic
horseradish	horse-rad-ish
hy-potha-la-mus	hy-po-thal-a-mus
ide-als	ideals
ideographs	ideo-graphs
id-iosyn-crasy	idio-syn-crasy
ig-niter	ig-nit-er
ig-n-i-tor	ig-ni-tor
ig-nores-paces	\ignore-spaces
impedances	im-ped-ances
in-finitely	in-fin-ite-ly
in-finites-i-mal	in-fin-i-tes-i-mal
in-fras-truc-ture	in-fra-struc-ture
in-ter-dis-ci-plinary	in-ter-dis-ci-pli-nary
inu-tile	in-utile
inu-til-ity	in-util-i-ty
ir-re-vo-ca-ble	ir-rev-o-ca-ble
itinerary(ies)	itin-er-ary(ies)
jeremi-ads	je-re-mi-ads
keystroke	key-stroke
kil-ning	kiln-ing
la-ciest	lac-i-est
lamentable	lam-en-ta-ble
land-sca-per	land-scap-er
larceny(ist)	lar-ce-ny(-ist)
lifes-pan	life-span
lightweight	light-weight
limousines	lim-ou-sines
linebacker	line-backer
lines-pac-ing	\line-spacing
lithographed	lith-o-graphed
lithographs	lith-o-graphs
lobotomy(ize)	lo-bot-omy(-ize)
lo-ges	loges
macroe-co-nomics	macro-eco-nomics
malapropism	mal-a-prop-ism
manuscript	man-u-script
marginal	mar-gin-al
mat-tes	mattes
med-i-caid	med-ic-aid
mediocre	medi-ocre
medi-o-crities	medi-oc-ri-ties
me-galith	mega-lith
metabolic	meta-bol-ic
metabolism	me-tab-o-lism
met-a-lan-guage	meta-lan-guage
metropo-lis(es)	me-trop-o-lis(es)
metropoli-tan	met-ro-pol-i-tan
mi-croe-co-nomics	micro-eco-nomics
mi-crofiche	mi-cro-fiche
mil-lage	mill-age
milliliter	mil-li-liter
mimeographed	mimeo-graphed
mimeographs	mimeo-graphs
mimi-cries	mim-ic-ries
mi-nis	min-is
min-uter(est)	mi-nut-er(-est)
mis-chievously	mis-chie-vous-ly
mis-ers	mi-sers
mis-ogamy	mi-sog-a-my
mod-elling	mod-el-ling
molecule	mol-e-cule
monar-chs	mon-archs
mon-eylen-der	money-len-der
monochrome	mono-chrome
mo-noen-er-getic	mono-en-er-getic
monoid	mon-oid
monopole	mono-pole
monopoly	mo-nop-oly
monos-pline	mono-spline
monos-trofic	mono-strofic
mono-tonies	mo-not-o-nies
monotonous	mo-not-o-nous
mo-ro-nism	mo-ron-ism
mosquito	mos-qui-to
mu-d-room	mud-room
mul-ti-faceted	mul-ti-fac-eted
mul-ti-pli-ca-ble	mul-ti-plic-able
mul-tiuser	multi-user (better with explicit hyphen)
ne-ofields	neo-fields
newslet-ter	news-let-ter
none-mer-gency	non-emer-gency
nonequiv-ari-ance	non-equi-vari-ance
noneu-clidean	non-euclid-ean
non-i-so-mor-phic	non-iso-mor-phic
nonpseu-do-com-pact	non-pseudo-com-pact
non-s-mooth	non-smooth
nonuni-form(ly)	non-uni-form(ly)
nore-pinephrine	nor-ep-i-neph-rine
nutcracker	nut-crack-er
oer-st-eds	oer-steds
oligopolist	oli-gop-o-list
oligopoly(ies)	oli-gop-oly(ies)
operand(s)	op-er-and(s)
orangutan	orang-utan
or-thodon-tist	or-tho-don-tist
or-thok-er-a-tol-ogy	or-tho-ker-a-tol-ogy
or-thoni-tro-toluene	ortho-nitro-toluene (or-tho-ni-tro-tol-u-ene)
overview	over-view
ox-i-dic	ox-id-ic
painlessly	pain-less-ly
pal-mate	palmate
parabola	par-a-bola
parabolic	par-a-bol-ic
paraboloid	pa-rab-o-loid
paradigm	par-a-digm
parachute	para-chute
paradimethyl-ben-zene	para-di-methyl-benzene (para-di-meth-yl-ben-zene)

paraflu-o-ro-toluene	para-fluoro-toluene (para-flu-o-ro-tol-u-ene)
para-g-ra-pher	para-graph-er
par-ale-gal	para-le-gal
par-al-lelism	par-al-lel-ism
para-m-ag-netism	para-mag-net-ism
paramedic	para-medic
param-ethy-lanisole	para-methyl-anisole (para-meth-yl-an-is-ole)
parametrize	pa-ram-e-trize
paramil-i-tary	para-mil-i-tary
paramount	para-mount
pathogenic	path-o-gen-ic
pee-vish(ness)	peev-ish(-ness)
pen-tagon	pen-ta-gon
petroleum	pe-tro-le-um
phe-nomenon	phe-nom-e-non
philatelist	phi-lat-e-list
phos-pho-ric	phos-phor-ic
pi-cador	pic-a-dor
pi-ran-has	pi-ra-nhas
pla-ca-ble	placa-ble
plea-sance	pleas-ance
poltergeist	pol-ter-geist
polyene	poly-ene
polyethy-lene	poly-eth-yl-ene
polygamist(s)	po-lyg-a-mist(s)
poly-go-niza-tion	polyg-on-i-za-tion
polyphonous	po-lyph-o-nous
polystyrene	poly-styrene
pomegranate	pome-gran-ate
poroe-las-tic	poro-elas-tic
postam-ble	post-am-ble
postscript	post-script
pos-tu-ral	pos-tur-al
pream-ble	pre-am-ble
preloaded	pre-loaded
prepar-ing	pre-par-ing
pre-pro-ces-sor	pre-proces-sor
pre-s-plit-ting	\pre-split-ting
priestesses	priest-esses
pro-ce-du-ral	pro-ce-dur-al
pro-cess	process
procu-rance	pro-cur-ance
pro-ge-nies	prog-e-nies
progeny	prog-e-ny
pro-hibitive(ly)	pro-hib-i-tive(-ly)
prosci-utto	pro-sciut-to
protestor(s)	pro-test-er(s)
protestor(s)	pro-tes-tor(s)
pro-to-ty-pal	pro-to-typ-al
pseu-dod-if-fer-en-tial	pseu-do-dif-fer-en-tial
pseud-ofi-nite	pseu-do-fi-nite
pseud-ofinitely	pseu-do-fi-nite-ly
pseud-o-forces	pseu-do-forces
pseudonym	pseu-do-nym
pseu-doword	pseu-do-word
psychedelic	psy-che-del-ic
psy-chs	psychs
pubescence	pu-bes-cence
quadrat-ics	qua-drat-ics
quadra-ture	quad-ra-ture
quadriplegic	quad-ri-pleg-ic
quain-ter(est)	quaint-er(est)
quasiequiv-a-lence	qua-si-equiv-a-lence or quasi-
quasi-hy-ponor-mal	qua-si-hy-po-nor-mal
quasir-ad-i-cal	qua-si-rad-i-cal
quasiresid-ual	qua-si-resid-ual
qua-sis-mooth	qua-si-smooth
qua-sis-ta-tion-ary	qua-si-sta-tion-ary
qu-a-sito-pos	qua-si-topos
qu-a-si-tri-an-gu-lar	qua-si-tri-an-gu-lar
quintessence	quin-tes-sence
quintessen-tial	quin-tes-sen-tial
rab-bi-try	rab-bit-ry
ra-dio-g-ra-phy	ra-di-og-ra-phy
raf-f-ish(ly)	raff-ish(-ly)
ramshackle	ram-shackle
ravenous	rav-en-ous
re-ar-range-ment	re-arrange-ment
re-ciproc-i-ties	rec-i-proc-i-ties
reci-procity	rec-i-proc-i-ty
rect-an-gle	rec-tan-gle
ree-cho	re-echo
restorable	re-stor-able
ret-ri-bu-tion(s)	ret-ri-bu-tion(s)
retrofit(ted)	retro-fit(-ted)
rhinoceros	rhi-noc-er-os
righ-teous(ness)	right-eous(-ness)
ringleader	ring-leader
robot	ro-bot
robotics	ro-bot-ics
roundtable	round-table
salesclerk	sales-clerk
salescle-rks	sales-clerks
saleswoman(en)	sales-woman(en)
salmonella	sal-mo-nel-la
sarsaparilla	sar-sa-par-il-la
sauerkraut	sauer-kraut
sca-to-log-i-cal	scat-o-log-i-cal
schedul-ing	sched-ul-ing
schizophrenic	schiz-o-phrenic
schnauzer	schnau-zer
schoolchild(ren)	school-child(-ren)
schoolteacher	school-teacher
scy-thing	scyth-ing
semaphore	sem-a-phore
semester	se-mes-ter
semidef-i-nite	semi-def-i-nite
semi-ho-mo-th-etic	semi-ho-mo-thet-ic
semir-ing	semi-ring
semiskilled	semi-skilled
seroepi-demi-o-log-i-cal	sero-epi-de-mi-o-log-i-cal
ser-vomech-a-nism	ser-vo-mech-anism
setup	set-up
severely	se-vere-ly
sha-peable	shape-able
shoestring	shoe-string
sidestep	side-step
sideswipe	side-swipe
skyscraper	sky-scraper
smokestack	smoke-stack
snorke-l-ing	snor-kel-ing
solenoid	so-le-noid
so-lute(s)	solute(s)
sovereign	sov-er-eign
spaces	spa-ces

specious	spe-cious
spelunker	spe-lunk-er
spendthrift	spend-thrift
spheroid(al)	spher-oid(al)
sph-inges	sphin-ges
spi-cily	spic-i-ly
spinors	spin-ors
spokeswoman(en)	spokes-woman(en)
sportscast	sports-cast
sportively	spor-tive-ly
sportswear	sports-wear
sportswriter	sports-writer
sprightlier	spright-lier
squeamish	squea-mish
stan-dalone	stand-alone
startling(ly)	star-tling(ly)
statis-tics	sta-tis-tics
stealthily	stealth-ily
steeplechase	steeple-chase
stochas-tic	sto-chas-tic
straight-est	straight-est
strangeness	strange-ness
stratagem	strat-a-gem
stretchier	stretch-i-er
stronghold	strong-hold
stupi-der(est)	stu-pid-er(est)
summable	sum-ma-ble
su-perego	super-ego
su-pere-gos	super-egos
supremacist	su-prema-cist
surveil-lance	sur-veil-lance
swim-m-ingly	swim-ming-ly
symp-tomatic	symp-to-matic
syn-chromesh	syn-chro-mesh
syn-chronous	syn-chro-nous
syn-chrotron	syn-chro-tron
talkative	talk-a-tive
tapestry(ies)	ta-pes-try(ies)
tarpaulin	tar-pau-lin
tele-g-ra-pher	te-leg-ra-pher
telekinetic	tele-ki-net-ic
teler-obotics	tele-ro-bot-ics
testbed	test-bed
tha-la-mus	thal-a-mus
ther-moe-las-tic	ther-mo-elas-tic
times-tamp	time-stamp
toolkit	tool-kit
to-po-graph-i-cal	topo-graph-i-cal
to-ques	toques
traitorous	trai-tor-ous
transceiver	trans-ceiver
transgress	trans-gress
transver-sal(s)	trans-ver-sal(s)
transvestite	trans-ves-tite
traversable	tra-vers-a-ble
traver-sal(s)	tra-ver-sal(s)
treacheries	treach-eries
treach-ery	treach-ery
troubadour	trou-ba-dour
turnaround	turn-around
ty-pal	typ-al
unattached	un-at-tached
unerringly	un-err-ing-ly
un-friendly(ier)	un-friend-ly(i-er)
va-guer	vaguer
vaudeville	vaude-ville
vi-cars	vic-ars
vil-lai-ness	vil-lain-ess
viviparous	vi-vip-a-rous
voiceprint	voice-print
vs-pace	\vspace
wallflower	wall-flower
wastew-a-ter	waste-water
waveg-uide	wave-guide
we-b-like	web-like
weeknight	week-night
wheelchair	wheel-chair
whichever	which-ever
whitesided	white-sided
whites-pace	white-space
widespread	wide-spread
wingspread	wing-spread
witchcraft	witch-craft
workhorse	work-horse
wraparound	wrap-around
wretched(ly)	wretch-ed(-ly)
yesteryear	yes-ter-year

Names and non-English words used in English text

al-ge-brais-che	al-ge-brai-sche
Al-legheny	Al-le-ghe-ny
Arkansas	Ar-kan-sas
au-toma-tisierter	auto-mati-sier-ter
Be-di-enung	Be-die-nung
Brow-n-ian	Brown-ian
Columbia	Co-lum-bia
Czechoslo-vakia	Czecho-slo-va-kia
Di-jk-stra	Dijk-stra
dy-namis-che	dy-na-mi-sche
En-glish	Eng-lish
Eu-le-rian	Euler-ian
Febru-ary	Feb-ru-ary
Festschrift	Fest-schrift
Florida	Flor-i-da
Forschungsin-sti-tut	For-schungs-in-sti-tut
funk-t-sional	funk-tsional
Gaus-sian	Gauss-ian
Greif-swald	Greifs-wald
Grothendieck	Grothen-dieck
Grundlehren	Grund-leh-ren
Hamil-to-nian	Hamil-ton-ian
Her-mi-tian	Her-mit-ian
Hi-bbs	Hibbs
Jan-uary	Jan-u-ary
Japanese	Japan-ese
Kadomt-sev	Kad-om-tsev
Kansas	Kan-sas
Karl-sruhe	Karls-ruhe
Ko-rteweg	Kor-te-weg
Leg-en-dre	Le-gendre
Le-ices-ter	Leices-ter
Lip-s-chitz(ian)	Lip-schitz(-ian)
Louisiana	Lou-i-si-ana
Manch-ester	Man-ches-ter
Marko-vian	Mar-kov-ian
Mas-sachusetts	Mass-a-chu-setts

Min-nesota	Min-ne-sota
Ni-jmegen	Nij-me-gen
Noethe-rian	Noe-ther-ian
No-ord-wi-jk-er-hout	Noord-wijker-hout
Novem-ber	No-vem-ber
Poincare	Poin-care
Po-ten-tial-gle-ichung	Po-ten-tial-glei-chung
rathskeller	raths-kel-ler
Rie-man-nian	Rie-mann-ian
Ry-d-berg	Ryd-berg
schot-tis-che	schot-tische
Schrodinger	Schro-ding-er
Schwabacher	Schwa-ba-cher
Schwarzschild	Schwarz-schild
Septem-ber	Sep-tem-ber
Stokess-che	Stokes-sche
Susque-hanna	Sus-que-han-na
tech-nis-che	tech-ni-sche
Ten-nessee	Ten-nes-see
ve-r-all-ge-mein-erte	ver-all-ge-mein-erte
Verteilun-gen	Ver-tei-lun-gen
Wahrschein-lichkeit-s-the-o-rie	Wahr-schein-lich-keits-the-o-rie
Werthe-rian	Wer-ther-ian
Winch-ester	Win-ches-ter
Yingy-ong Shuxue Jisuan	Ying-yong Shu-xue Ji-suan
Zeitschrift	Zeit-schrift

Literate Programming

Errata: Literate Programming, A Practitioner's View *TUGboat* 13, no. 3, pp. 261–268

Bart Childs

The address `lyman.pppl.princeton.edu` should have read `lyman.pppl.gov` (a careless error on my part). The address `csseq.cs.tamu.edu` no longer permits anonymous ftp. Due to some network breakins from different places, extensive local network changes are being done and relevant sources will be placed on `ftp.cs.tamu.edu` no later than January 1993. The author will e-mail any desired sources in the meantime.

◇ Bart Childs
Texas A&M University
Department of Computer Science
College Station, TX 77843-3112
`bart@cs.tamu.edu`

Philology

Hyphenation patterns for ancient Greek and Latin

Yannis Haralambous

The amount of compound words in ancient Greek makes its hyphenation by computer a quite difficult task; it is impossible to predict all combinations of words. To be efficient, a set of patterns must be accessible to the final user; a scholar must be able to add patterns, according to new words he/she encounters. Use of TEX's `\hyphenation` primitive is not appropriate since most Greek words are declinable: for each word one would have to add a dozen hyphenation exceptions.

After a short introduction to the concept of hyphenation by TEX the author presents a method for hyphenation of ancient Greek. Using this method, he compiled a list of patterns out of a dictionary [Bai] of 50,000 words. These patterns are presented in a comprehensible format, in a way that scholars can easily determine the patterns that have to be added, to solve specific hyphenation problems.

The same approach is applied to Latin. A list of patterns has been compiled out of a dictionary [B–C]. The size of this list is very small compared to the one of ancient Greek patterns, although Latin also uses compound words.

Finally examples of hyphenated classical texts are given.

1 What do I have to know about hyphenation?

When TEX creates a format like `Plain` or `LaTeX` it also reads information from a file called `hyphen.tex` (or `UShyphen.tex`, or `FRhyphen.tex` and so forth) which contains the *hyphenation patterns* for a specific language. These are clusters consisting of letters separated by digits, like `x1y2z`. The idea is the following:

- if your set of patterns is empty, there is no hyphenation at all.
- if you have a pattern `x1y` then on every occurrence of the cluster "xy", hyphenation "x-y" will be possible. If the pattern is `x1yzw`, then the pair of letters "xy" will be hyphenated only when followed by "zw".

- if there is a pattern `x1y` and a pattern `x2yabc`, then the pair "xy" will be hyphenated, except when it is followed by "abc". So the digit 2 indicates an exception to the rule "*separate x and y*".
- the same holds for greater numbers: 3 will be an exception of patterns with number 2, and so on. You can now read [DEK], pp. 449–451, for more details on TEX's hyphenation algorithm.
- a dot in front of (or behind) a pattern specifies that the latter is valid only at the beginning (or the end) of a word. In this way, for example, `.xy2z.` will be applied *only* to the word 'xyz'.

Despite the existence of some fundamental rules, hyphenation of a particular language can be very complicated, especially when it depends on etymological criteria. There are two ways to handle this complexity: one can investigate the hidden mechanisms of hyphenation and make patterns correspond to the analytical steps of manual hyphenation; or one can use a "pattern generator" like PATGEN on a sufficiently representative set of already hyphenated words. The choice of the method depends on the nature of the language and on the size of the available set of hyphenated words: theoretically one could create a file containing *all* words of a particular language in hyphenated form; the pattern generator would then give an *exhaustive* set of patterns. Since it is more probable to have partial sets of words, the "pattern generator" will only produce more or less good approximations. For ancient Greek and Latin, the author has chosen the first method.

2 Why is hyphenation of ancient Greek so difficult for a computer?

As mentioned in *TUGboat* 11, no. 1, since 1990 patterns have existed for modern Greek (made by the author). What then makes hyphenation of ancient Greek so different? Why is it so difficult?

First of all, because of compound words. The difficulty lies in the fact that components are often altered when composed: ἐπὶ and αἰνῶ gives ἐπαινῶ (not ἐπιαινῶ) while ἐπὶ composed with βάλλω produces ἐπιβάλλω. Could we hence hyphenate always ἐπ-α and ἐπι-? well, actually not, because there is also the case of ἐπὶ + ἰάλλω = ἐπιάλλω which is hyphenated ἐπ-ιάλλω. Then perhaps we could produce patterns for *roots*, like -αιν, to insure hyphenation of ἔπ-αινος, κατ-αίνεσις, ψευδ-αινῶ and so forth? No, because this would produce wrong hyphenation of ἀ-σθμαί-νω, ση-μαί-νω, ὑ-φαί-νω...

A second problem is declension and diacritics. While the latter are provided to facilitate comprehension of a word, they rather obstruct the work of the computer. Because for the computer α, ά, ὰ, ᾶ, ἀ, ἁ, etc., are *distinct* entities. So ἐπεὶ and ἐπεί are to be treated as entirely distinct words. In most languages declension affects only endings of words: здание, здания, зданию, здание, зданием, здании and so forth. In Greek, the root of the word remains mostly the same (except in cases such as ὁ ἀνήρ, τοῦ ἀνδρὸς) but the accent migrates: ὁ ἄνθρωπος, τοῦ ἀνθρώπου. It follows that when creating patterns, one must consider all possible diacritics and their positions.

To illustrate this, here is an example of two words which look very similar but are hyphenated in two different ways:

ἄφ-ορρος (from ἄφ and ὄρνυμι)
and
ἀφό-ρροος (or ἀφόρρους, from ἄφ and ῥέω).

We will decline them and try to extract the necessary patterns so that TEX can correctly hyphenate them, in all cases[1]:

ὁ	ἄφορρος	ὁ	ἀφόρροος
τοῦ	ἀφόρρου	τοῦ	ἀφορρόου
τῷ	ἀφόρρῳ	τῷ	ἀφορρόῳ
τὸν	ἄφορρον	τὸν	ἀφόρροον
ὦ	ἄφορρε	ὦ	ἀφόρροε
οἱ	ἄφορροι	οἱ	ἀφόρροοι
τῶν	ἀφόρρων	τῶν	ἀφορρόων
τοῖς	ἀφόρροις	τοῖς	ἀφορρόοις
τοὺς	ἀφόρρους	τοὺς	ἀφορρόους
ὦ	ἄφορροι	ὦ	ἀφόρροοι

For the word ἄφορρος it would be enough to introduce patterns ἄφ-ορρ and ἀφ-όρρ. In this case, the word ἀφόρροος would be wrongly hyphenated. To distinguish this word we must include an o at the end of the pattern, and hence introduce the pattern ἀφό-ρρο. This would again change the hyphenation of the genitive case of ἄφορρος, namely ἀφόρρου. For this reason we will rather use the pattern ἀφό-ρροο. This has the advantage of not interfering with the hyphenation of ἄφορρος, but cannot be applied to the vocative case of ἀφόρροος. For this, we need a second pattern ἀφό-ρροε, which is actually the whole word. In the remaining cases of ἀφόρροος, the accent is on the penultimate syllable ρρο. But the word ἄφορρος is never accented on that syllable, so that an unaccented pattern like ἀφο-ρρ would apply *only*

[1] The reader should excuse the unusual order of cases: nominative, genitive, dative, accusative, vocative. This order is used in education and scholarly literature, in Greece.

to the genitive, dative and plural accusative cases of ἀφόρροος.

Using this method, we found five patterns:
ἄφ-ορρ, ἀφ-όρρ, ἀφό-ρροο, ἀφό-ρροε, ἀφο-ρρ
The reader can verify that with these, all cases of both words are correctly hyphenated.

The example illustrates another fact: when writing down new patterns, one must constantly compare the newly hyphenated words with the ones which produced the previous patterns, so that conflicts can be avoided.

Finally there are some rare cases where identical words have different meanings and different hyphenations: ἄ-νοστος from ἀ and νόστος, ἄν-οστος from ἀν and ὄζω; ἐν-ετὸς from verb ἐν-ίημι and ἐ-νετὸς (Venitian).

3 The fundamental rules of ancient Greek hyphenation, and the corresponding patterns

The Chicago Manual of Style ([Chi], 9.130) asserts that: IN [ancient] GREEK, WORD DIVISION FOLLOWS RULES THAT ARE STRAIGHTFORWARD AND FAIRLY EASY TO APPLY.

Here are the rules following this quotation in [Chi], and the necessary patterns ($v_n, n \geq 1$ will be vowels and $c_n, n \geq 1$ consonants):

1. WHEN A SINGLE CONSONANT OCCURS BETWEEN TWO VOWELS, DIVIDE BEFORE THE CONSONANT: v_1–cv_2. The necessary patterns will be α1β, α1γ, α1δ, ... ω1ψ where vowels are taken with all possible combinations of accents, spirits, diæresis and subscript iota. These patterns cover also the case of a vowel followed by more than one consonant; this feature will be useful in rule 4.
2. IF A CONSONANT IS DOUBLED, OR IF A MUTE IS FOLLOWED BY ITS CORRESPONDING ASPIRATE, DIVIDE AFTER THE FIRST CONSONANT: v_1c_1–c_2v_2 for $c_1 = c_2$, or $(c_1, c_2) \in \{(\pi, \varphi), (\tau, \theta), (\kappa, \chi), (\gamma, \chi)\}$. The patterns will be 2β1β, 2γ1γ ... 2ψ1ψ for the first part of the rule, and 2π1φ, 2τ1θ, 2κ1γ, 2γ1χ for the second. For grammatical reasons, it would be best to exclude 2ρ1ρ (and 2ῥ1ῥ) from these patterns.
3. IF THE COMBINATION OF TWO OR MORE CONSONANTS BEGINS WITH A LIQUID OR A NASAL, DIVIDE AFTER THE LIQUID OR NASAL [although not stated in the rule, it follows from the examples given in [Chi] that two consecutive nasals should not be separated]: v_1c_1–$c_2 \ldots c_nv_2$, for $c_1 \in \{\lambda, \rho, \mu, \nu\}$ but v_1–$c_1 \ldots c_nv_2$ if $(c_1, c_2) = (\mu, \nu)$. The required patterns are 2λ1β, 2λ1γ ... 2ν1ψ (except of course 2λ1λ, 2ρ1ρ, 2μ1μ, 2ν1ν which have been taken into account in rule 2), and 2μ1ν.
4. THE DIVISION COMES BEFORE ALL OTHER COMBINATIONS OF TWO OR MORE CONSONANTS. We do not need any additional patterns to handle this rule; for example in the word ἄστρον, the cluster ἄσ will be hyphenated because of rule 1, and clusters στ, τρ, ρο will not be hyphenated, because they do not appear in any previous pattern list. Theoretically, a problem could occur in the case of a combination of 3 or more consonants *containing* a cluster mentioned in rules 2 or 3. But this is highly improbable and should be taken as an exception.
5. COMPOUND WORDS ARE DIVIDED INTO THEIR ORIGINAL PARTS; WITHIN EACH PART THE FOREGOING RULES APPLY. The patterns needed to fulfill this rule will be discussed in next section.

It might be interesting to point out that the rules specified by the *Academy of Athens* in 1939 ([Aca], Ἔγκλισις τόνου καὶ συλλαβισμὸς) are slightly different from the ones above. According to this set of rules, compound words are separated, except when an eclipse has occurred: παρέχω is hyphenated as if it was a simple word: πα-ρέ-χω, instead of παρ-έχω as suggested in [Chi]. Following the rules of the Academy of Athens would result in a completely different set of patterns, since the eclipse phenomenon occurs very often.

The patterns we have introduced thus far are not sufficient for fundamental hyphenation. We still must introduce two families of patterns:

- Pairs of vowels are to be separated, except in the case of diphthongs. This leads to patterns α1α, α1ε, α1η, α1ϊ ... ω1ω including all diacriticized vowels.
- In Greek, the smallest part of a word remaining on a line is a syllable. This may shock a TEX user, but hyphenations like ἀ-προσάρμοστος and ἐπιδεικνύ-ω are allowed, and can frequently be found in books. The problem is that by setting `\lefthyphenmin` and `\righthyphenmin` equal to 1, one can eventually separate single consonants, which do not form syllables: δῶρο-ν is TEXnically allowed, since `\righthyphenmin=1`, but should be avoided. For this we just have to introduce patterns 2β., 2δ. ... 2ψ. [or even 6β., 6δ. ... 6ψ. to be sure that no forthcoming exception will affect them].

What remains now are patterns concerning separation of compound words, according to rule 5. These are described in the following section.

4 Hyphenation of ancient Greek compound words

> *No puede combinar unos caracteres* ***dhcmrlchtdj*** *que la divina Biblioteca no haya previsto y que en alguna de sus lenguas secretas no encierren un terrible sentido...*

writes J. L. Borges in the "Biblioteca de Babel"; the situation is similar for ancient (or modern) Greek compound words. By combining words one can easily exceed Mark Twain's

> "Mekkamuselmannenmassenmenchenmördermohrenmuttermarmormonumentenmachen",

given in [DEK], p. 451. As already pointed out, one could make patterns out of all possible roots, so that any possible combination of them is correctly hyphenated. The problem is, though, that the combinations of letters forming these roots can also occur inside single words, and only by their meaning can one decide if a particular root is present in a word: πεντήρης is formed by πεντ- and -ηρ- (from ἄρω). The words τριήρης, μονήρης, τειχήρης, all contain the same root -ηρ-. But introducing a pattern 3η4ρ3 would cause tremendous problems, since thousands of words contain the cluster ηρ, hyphenated as η-ρ: τεκμή-ριον, στή-ριγμα, πη-ρόω and so forth. The fact is that such patterns should be avoided, or introduced only after extensive investigation.

The author has chosen a different method. Instead of introducing patterns for roots of words, only *beginnings of words* are taken into account. In this way, when writing for example .ἐ4ν3 one can be sure that only words beginning with the prefix ἐν will be affected. Exceptions to this rule can easily be found by consulting the dictionary. This method is very effective in hyphenating the most frequent words, but fails when new compound words are to be hyphenated: ἐπ-αινῶ and κατ-αινῶ will be correctly hyphenated (since patterns ἐπ- and κατ-αι are included in the list) but not an eventual κατ-επ-αινῶ, unless of course the user adds a new pattern κατ-επ- to the list in section 5.

Another advantage of this method is the fact that, contrary to most pattern lists, such a list is easily comprehensible, and hence can be completed easily by the final user himself (who does not have to be a TEX-guru). This comes from the fact that — except for the fundamental patterns explained in the previous section — all patterns are beginnings of words. By checking in the list, one can instantly verify why a specific word is hyphenated in a particular way; once this is understood, one can add the necessary pattern(s) to remedy to the situation. This is highly inadvisable for a pattern file created by a "pattern generator", where a simple change can have very strange and obscure results (until now pattern files always started with the most categorical phrase `[NOT TO BE CHANGED IN ANY WAY!]`).

To facilitate even more the comprehension of this list of patterns, the author has chosen to present it in a very special format. Here is a sample excerpt of the pattern list which the reader can find in next section:

αὐτό-σσ*(υτος)*, αὐτο-σσ*(ύτου)*
ἄφ-, ἀφ-
- ♦ ἄ-φαλ*(ος)*, ἀ-φάλ*(ου)*
  - ♦ ἀφ-άλλ*(ομαι)*
- ♦ ἀ-φάν*(εια)*, ἀ-φαν*(ής)*
  - ♦ ἀφ-ανδ*(άνω)*

The words αὐτό-σσ*(υτος)* and αὐτο-σσ(ύτου) constitute one entry. In both cases, the patterns themselves are given in straight characters: αὐτό-σσ and αὐτο-σσ. The slanted endings between parentheses are *just possible examples*, which justify the existence of these patterns. It is important that the reader realizes that *these examples are not unique* but just indicative. So only what is written in straight letters will appear as a pattern.

The entry ἄφ-, ἀφ- is called a *rule*. Because of its frequency, the author has preferred not to give any example, and to present it "as a general rule". The symbol ♦ introduces exceptions, and exceptions to exceptions. The difference is shown by indentation. In this case, ἀ-φάλ is an exception to ἀφ-, and ἀφάλου (genitive case of ἄφαλος) is an example of a word starting with this cluster. ἀφ-άλλ is an exception to ἀ-φάλ, and ἀφάλλομαι a possible example. Same thing for ἀφ-, ἀ-φάν, ἀφ-ανδ.

Let's take a look to the concrete realization of these patterns: since ἀφ- is an exception to the fundamental hyphenation rules, the pattern can for example be .ἀ2φ1; the exception ἀ-φάλ could be expressed as "*if* ἀφ *is followed by* άλ*, do not cut between* φ *and* ά*, but do cut between* ἀ *and* φ". This leads us to the pattern .ἀ3φ2άλ . The reader can verify that the further exception ἀφ-άλλ requires a pattern like .ἀ4φ3άλλ .

In the next section, the complete list of patterns extracted from [Bai] is presented, in the format explained above.

5 Patterns for ancient Greek compound words

ἀγαπ-ήν(ωρ), ἀγαπ-ην(όρων)
ἀγελ-άρ(χης), ἀγελ-αρ(χῶν)
ἄγρ-υ(πνος), ἀγρ-ύ(πνου)
ἀγχ-ώ(μαλος), ἀγχ-ω(μάλου)
ἀγων-ά(ρχης), ἀγων-α(ρχῶν)
αἰχμ-άλ(ωτος), αἰχμ-αλ(ώτου)
ἀκρ-ών(υχος), ἀκρ-ων(ύχου)
ἀκρ-ώρ(εια), ἀκρ-ωρ(είας)
ἀλέξ-αν(δρος), ἀλεξ-άν(δρου), ἀλεξ-αν(δρινὸς)
ἄμ-α(ξα), ἀμ-ά(ξης), ἀμ-α(ξῶν)
ἀμβλ-ω(πὸς)
ἀμπ-έχ(ω), ἀμφ-έξ(ω)
♦ ἀμ-πε-χό(νη)
ἀμπ-ί(σχω), ἀμπ-ι(σχνέομαι)
ἀμφ-αγ(απάζω)
ἀμφ-αρ(αβέω)
ἀμφ-έπ(ω)
ἀμφέρχ(ομαι)
ἀμφ-ή(κης), ἀμφ-η(κῶν)
ἀμφ-ιάχ(ω)
ἀμφ-ιζά(νω)
ἀμφίσ-β(αινα), ἀμφισ-β(ατέω)
ἀμφ-ίστ(ημι), ἀμφ-ιστ(ήσω)
♦ ἀμφί-στο(μος), ἀμφι-στό(μου)
ἄμφ-ου(δις), ἀμφ-ού(δεως)
ἄμφ-ωτ(ος), ἀμφ-ώτ(ου), ἀμφ-ωτ(ίς)
ἄν-αγνο(ς), ἄν-αγνε, ἀν-άγνο(υ), ἀν-άγνῳ
ἀν-αγο(ρεύω)
ἀν-άγω, ἀν-αγώ(γιον), ἀν-αγω(γὴ)
ἀν-άδελ(φος), ἀν-αδέλ(φου)
ἀν-αεί(ρω)
ἀν-αθρέ(ω)
♦ ἀνα-θρέφ(ω)
ἄν-αι(μος), ἀν-αί(νομαι), ἀν-αῖ(σσω), ἀν-αι(μωτὶ)
ἀν-άκαν(θος), ἀν-ακάν(θου), ἀν-ακαν(θίνου)
ἀν-ακέο(μαι), ἀν-ακέε(σαι)
ἀν-ακο(ῦμαι)
ἀν-ακωχ(ὴ)
ἀν-αλαλ(άζω)
ἀν-άλγ(ητος), ἀν-αλγ(ὴς)
ἀν-αλήθ(ης), ἀν-αλῆθ(ες)
ἀν-αλί(σκω)
ἄν-αλκ(ις), ἀν-άλκ(ιδος), ἀν-αλκ(εία)
ἀν-αλλ(οίωτος)
ἄν-αλμ(ος), ἀν-άλμ(ου)
ἀν-αλό(ω)
♦ ἀνα-λόγ(ως)
ἄν-αλτ(ος), ἀν-άλτ(ου), ἀν-αλτ(έστερος)
ἀν-άλω(τος), ἀν-αλώ(του)
ἀν-αμάξ(ευτος), ἀν-αμαξ(εύτου)
ἀν-αμάρ(τητος), ἀν-αμαρ(τήτου)
ἀν-άμβ(ατος), ἀν-αμβ(άτου)
ἀν-αμφ(ίλογος)
ἄν-ανδ(ρος), ἀν-άνδ(ρου), ἀν-ανδ(ρία)
ἀν-αντ(ίρρητος)
ἄν-αντ(α), ἀν-άντ(ης)
ἀν-άξι(ος), ἀν-αξί(ου), ἀν-αξι(ότερος)
ἀν-απλό(ω)
ἀν-αποδε(ίκτως)
ἀν-απόδρ(αστος), ἀν-αποδρ(άστου)
ἀν-άποι(νος), ἀν-αποί(νου)
ἀν-απόλα(υστος), ἀν-απολα(ύστου)
ἀν-απόσ(τατος), ἀν-αποσ(τάτου)
ἀν-άπτ(ω), ἀν-άψ(ω)
ἄν-αρ(θρος), ἀν-άρ(θρου), ἀν-αρ(ίθμητος)
ἀν-άσκη(τος), ἀν-ασκή(του)
ἄν-ατο(ς), ἀν-άτο(υ), ἀν-άτω(ν)
♦ ἀνα-τολ(ὴ)
ἄν-αυ(δος), ἀν-αύ(γητος), ἀν-αυ(γήτου)
♦ ἄ-ναυς
ἀν-αφρόδ(ιτος), ἀν-αφροδ(ίτου)
ἀνδρ-άγ(ρια), ἀνδρ-αγ(αθέω)
ἀνδρ-αχ(θὴς)
ἀνδρ-είκ(ελος), ἀνδρ-εικ(έλου)
ἀνδρ-ηλ(άτης)
ἄν-ε, ἀν-έ, ἀν-ε
♦ ἄ-νευ
♦ ἀ-νέφε(λος), ἀ-νεφέ(λου)
♦ ἀ-νεψ(ιὰ)
ἄν-η, ἀν-ή, ἀν-ῆ, ἀν-η
♦ ἀ-νὴρ, ἀ-νὴρ
ἀνθ-
♦ ἄν-θ(ησις)
♦ ἄν-θεμ(ον), ἀν-θέμ(ιον), ἄν-θεμ(ίζομαι)
♦ ἀν-θέω, ἀν-θέε(ις), ἀν-θέο(μεν)
♦ ἀν-θῶ(μεν), ἀν-θεῖ(τε), ἀν-θοῦ(σι)
♦ ἀν-θήσ(εως), ἀν-θηρ(ὸς)
♦ ἀν-θί(ζω), ἀν-θι(νὸς)
♦ ἀνθό-(κροκος), ἀνθο-(κρόκου)
♦ ἀνθ-ομ(ολογοῦμαι)
♦ ἀνθ-οπ(λίζω)
♦ ἀνθ-ορ(μέω)
♦ ἀνθ-οσ(μία)
♦ ἀν-θρ(ακεὺς)
ἄν-ι, ἀν-ί, ἀν-ι
♦ ἀ-νίκη(τος), ἀ-νική(του)
♦ ἀ-νιπτό(πους)
♦ ἄ-νιπ(τος), ἀ-νίπτο(υ), ἀ-νίπτω(ν)
ἄν-ο, ἀν-ό, ἀν-ο
ἀ-νόη(τος), ἀ-νοή(του)
♦ ἄ-νομο(ς), ἀ-νόμο(υς), ἀ-νόμω(ν)
♦ ἄ-νοο(ς), ἀ-νόο(υ)
♦ ἄ-νοσ(ος), ἀ-νόσ(ου)
♦ ἀν-όσι(ος), ἀν-οσί(ου)
♦ ἀ-νουθ(έτητος)
ἀντ-ά, ἀντ-α
ἀντ-έ, ἀντ-ε
♦ ἀν-τείν(ω)
♦ ἀν-τέλλ(ω)
ἀντ-ή, ἀντ-η
ἀντ-ισό(ω)
ἀντ-ισχ(υρίζομαι)
ἀντ-ο, ἀντ-υ, ἀντ-ω
ἄν-υ, ἀν-ύ, ἀν-υ
♦ ἄ-νυμφ(ος), ἀ-νύμφ(ευτος), ἀ-νυμφ(εύτου)

♦ ἀ-νύω, ἀ-νύο(υν), ἀ-νύε(ις)
ἀν-ώ, ἀν-ω
♦ ἀνω-φερ(ὴς)
ἀπ-ά, ἀπ-α
♦ ἀ-παγή(ς), ἀ-παγὴ(ς), ἀ-παγο(ῦς), ἀ-παγῶ(ν)
♦ ἀ-παγέ(ς), ἀ-παγὲ(ς), ἀ-παγε(ῖ), ἀ-παγῆ
♦ ἀ-πάθ(εια), ἀ-παθ(είας)
♦ ἀπ-αθα(νατίζω)
♦ ἀ-πάλα(μνος), ἀ-παλά(μνου)
♦ ἀ-πάρ(θενος), ἀ-παρ(θένου)
♦ ἀπ-αρί(θμητος), ἀπ-αρι(θμέω)
♦ ἀπ-αρκ(έω)
♦ ἀπ-αρν(έομαι)
♦ ἀπ-αρτ(άω)
♦ ἀπ-αρύ(τω)
♦ ἀπ-άρχ(ω)
♦ ἀ-πάτω(ρ), ἀ-πάτο(ρος)
ἀπ-έδο(μαι), ἀπ-εδό(μεθα), ἀπ-έδε(ισθε)
ἀπ-εθί(ζω)
ἀπ-εικ(άζω)
ἀπ-είλ(ημα), ἀπ-ειλ(έω)
ἄπ-ει
♦ ἄ-πειρ(ος)
ἀπ-είπ(ομεν), ἀπ-εῖ(πον)
ἀπ-είργ(ω), ἀπ-ειργ(άθω)
ἀπ-έκ(δυνον), ἀπ-εκ(δύνω)
ἀπ-έλ(αυνον), ἀπ-ελ(αύνω)
♦ ἀ-πέλε(θρος)
ἀπ-έμ(εσσα), ἀπ-εμ(έω)
ἀπ-έν(επον), ἀπ-εν(έπω)
♦ ἀ-πένθ(ητος), ἀ-πενθ(ής)
ἀπ-έο(ικα)
ἀπ-έρασ(ις), ἀπ-εράσ(εως)
ἀπ-έργ(ω), ἀπ-εργ(άζομαι)
ἀπ-έρδ(ω)
ἀπερ-εί, ἀπερ-εὶ
ἀπ-ερε(ίδω)
ἀπ-ερύ(κω), ἀπ-ερυ(θριάω)
ἀπ-έρχ(ομαι), ἀπ-ερχ(όμεθα)
ἀπ-έρω(τος), ἀπ-ερώ(του), ἀπ-ερω(έω)
ἀπ-εσθ(έομαι)
ἀπ-ευθύ(νω)
ἀπ-ευν(άζω)
ἀπ-εύχ(ομαι), ἀπ-ευχ(όμεθα)
ἀπ-ευω(νίζω)
ἄπ-εφ(θος), ἀπ-έφ(θου), ἀπ-εφ(θέστερος)
ἀπ-έχ(θομαι), ἀπ-εχ(θαίρω)
ἀπ-ήγ(γελον)
ἀπ-ηλ(εγέως)
ἀπ-ήω(ρος), ἀπ-ηώ(ρου)
ἀπ-ιά(λλω)
ἀπ-ιπ(πόω)
ἀπ-ισ(όω)
♦ ἀ-πιστ(έω)
ἀπ-ίστ(ημι)
ἀπ-ίσχ(ω), ἀπ-ισχ(υρίζομαι)
ἄπ-οικ(ος), ἀπ-οίκ(ου), ἀπ-οικ(έω)
ἀπ-οιμ(ώζω)
ἀπ-όλλυ(μι), ἀπ-ολλύ(ω)
ἀπ-ολοφ(ύρομαι)
ἀπ-ομόρ(γνυμι)
ἀπ-ονίν(ημι)
ἀπ-ονυχ(ίζω)
ἀπ-οξύν(ω)
ἄπ-οπτ(ος), ἀπ-όπτ(ου)
♦ ἀπό-πτολ(ις)
ἄπ-ορ, ἀπ-όρ, ἀπ-ορ
♦ ἀ-πόρθ(ητος), ἀ-πορθ(ήτου)
♦ ἀπό-ρρ(ευσις), ἀπό-ῥ(ρευσις), ἀπο-ρρ(αθυμέω), ἀπο-ῥ(ραθυμέω)
♦ ἀ-πόρφυ(ρος), ἀ-πορφύ(ρου)
ἀπ-οσιό(ομαι), ἀπ-οσιο(ῦμαι)
ἄπ-ου, ἀπ-ού, ἀπ-ου
♦ ἄ-πους, ἄ-που
ἀπ-οχετ(εύω)
ἀπ-οχυρ(όω)
ἄπ-οψι(ς), ἀπ-όψε(ως)
ἀπ-ω(δέω)
ἀργυρ-ώ(νητος), ἀργυρ-ω(νήτου)
ἀρμ-άμ(αξα), ἀρμ-αμ(άξης)
ἀρχ-ηγέτ(ης), ἀρχ-ηγετ(ῶν)
ἀρχ-ηγό(ς), ἀρχ-ηγὸ(ς), ἀρχ-ηγο(ῦ)
ἀρχ-ηγέ, ἀρχ-ηγὲ, ἀρχ-ηγῶ(ν)
ἀρχ-ιε(ρεὺς)
ἀ-στεργ-άν(ωρ), ἀ-στεργ-αν(όρων)
αὐθ-
♦ αὐθι-(γενὴς)
αὖτ-, αὐτ-
♦ αὐτί-κα
♦ αὐτό-(βουλος), αὐτο-(βοεί)
♦ αὐτ-όπτ(ης)
♦ αὐτ-όρο(φος), αὐτ-ορό(φου)
αὐτό-σσ(υτος), αὐτο-σσ(ύτου)
ἄφ-, ἀφ-
♦ ἄ-φαλ(ος), ἀ-φάλ(ου)
♦ ἀφ-άλλ(ομαι)
♦ ἀ-φάν(εια), ἀ-φαν(ής)
♦ ἀφ-ανδ(άνω)
♦ ἄ-φαρ, ἀ-φάρ(μακτος), ἀ-φαρ(μάκτου)
♦ ἀφ-αρπ(άζω)
♦ ἄ-φατ(ος), ἀ-φάτ(ου)
♦ ἄ-φε(ρτος), ἀ-φέ(ρτου), ἀ-φε(γγής)
♦ ἀφ-εί(ργω)
♦ ἀφ-έλ(κω)
♦ ἄ-φθ(ιτος), ἀ-φθ(ίτου)
♦ ἀ-φίλ(ως), ἀ-φιλ(όκαλος)
♦ ἄ-φλ(οιος), ἀ-φλ(έγμαντος)
♦ ἄ-φοβ(ος), ἀ-φόβ(ητος), ἀ-φοβ(ήτου)
♦ ἀ-φόρ(ητος), ἀ-φορ(ήτου)
♦ ἄ-φορο(ς), ἀ-φόρο(υ), ἀ-φόρω(ν)
♦ ἄ-φρ(ουρος), ἀ-φρ(άσμων)
♦ ἄ-φυ(κτος), ἀ-φύ(λακτος), ἀ-φυ(ής)
♦ ἀφ-υβ(ρίζω)
♦ ἀφ-υπ(νίζω)
ἄψ-ορρ(ος), ἄψ-οῤ(ρος), ἀψ-όρρ(ου), ἀψ-όῤ(ρου)
♦ ἀψό-ρροο(ς), ἀψό-ρροε, ἀψο-ρρό(ου)
♦ ἀψό-ῤῥοο(ς), ἀψό-ῤῥοε, ἀψο-ῤῥό(ου)
βαλαν-ά(γρα), βαλαν-α(γρῶν)

βαλλ-α(χράδαι)
βητ-ά(ρμων), βητ-α(ρμώνων)
βλοσυρ-ώ(πιδος), βλοσυρ-ῶ(πις), βλοσυρ-ω(πίδων)
βοιωτ-ά(ρχης), βοιωτ-α(ρχία)
βόσ-π(ορος), βοσ-π(όρου)
βούλ-α(ρχος), βουλ-ά(ρχου)
γαμφ-(ῶνυξ)
γεροντ-αγ(ωγέω)
γιγαντ-ολ(έτης)
γλαυκ-ώπ(ιδος), γλαυκ-ῶπ(ις), γλαυκ-ωπ(ὸς)
γονυκαμφ-επ(ίκυρτος)
γονυκαλυσ-ά(γρυπνα)
γοργ-ώ(πιδος), γοργ-ῶ(πις), γοργ-ω(πὸς)
δεισ-ή(νωρ), δεισ-η(νόρων)
δευτερ-αγ(ωνιστὴς)
δεχ-ή(μερος), δεχ-η(μέρου)
δημ-αγ(ωγὸς)
δήμ-αρ(χος), δημ-άρ(χου), δημ-αρ(χικὸς)
δημ-ωφ(ελὴς)
δισ-θ(ανὴς)
δισσ-ά(ρχαι), δισσ-α(ρχῶν)
δισ-ύπ(ατος), δισ-υπ(άτου)
δισ-χ(ίλιοι)
δουρ-η(νεκὴς)
δύσ-, δυσ-
♦ δύ-σις, δύ-σε(ως), δύ-σιν
δωδεκάδ-αρ(χος), δωδεκαδ-άρ(χου)
δωδέκ-αρ(χος), δωδεκ-άρ(χου), δωδεκ-έ(της)
εἰκόσ-ο(ρος), εἰκοσ-ό(ρου)
εἰλ-άρ(χης), εἰλ-αρ(χῶν)
εἶσ-, εἰσ-
♦ εἶ-σω
ἐκ-εχ(ειρία)
ἑλίκ-ω(ψ), ἑλικ-ώ(πιδος), ἑλικ-ῶ(πις), ἑλικ-ω(πίδῳ)
ἔν-, ἐν-
♦ ἐντεῦθ(εν), ἐντευθ(εν᾽)
♦ ἐνταῦθ(α), ἐνταυθ(οῖ)
ἐξ-
ἔπ-, ἐπ-
♦ ἔ-παθο(ν), ἔ-παθα(ς), ἔ-παθε(ς)
♦ ἐ-πάθο(μεν), ἐ)πάθα(τε)
♦ **ἔπι-, ἐπί-, ἐπι-**
♦ ἐπ-ιάλλ(ω)
♦ ἐπ-ιαύ(ω)
♦ ἐπ-ιάχ(ω)
♦ ἐπ-ίη(μι)
♦ ἐπ-ιθύν(ω)
♦ ἐπ-ιλλ(ίζω)
♦ ἐπ-ίσταμ(αι), ἐπ-ίστασ(αι), ἐπ-ίστατ(αι)
♦ ἐπ-ιστάμ(εθα), ἐπ-ίσταν(ται)
♦ ἐπ-ίστι(ος), ἐπ-ιστί(ου)
♦ ἐπ-ιωγ(ὴ)
ἑρμ-αφ(ρόδιτος)
ἐρ-ράπ(τω)
ἔρ-ρυθ(μος), ἐρ-ρύθ(μου)
ἐρυσ-άρ(ματος), ἐρυσ-αρ(μάτου)
ἐσ-
ἑτερ-α(λκὴς)
ἑτερ-ή(μερος)
ἑτερ-όφθ(αλμος), ἑτερ-οφθ(άλμου)
ἔφ-, ἐφ-
ζευγ-η(λάτης)
ἤν-οψ, ἤν-οπ(ος)
θέσ-φ(ατος), θεσ-φ(άτου)
θιασ-ά(ρχης)
θυμ-α(λγὴς)
θυμ-η(δὴς)
θυσσ-κό(ος)
θυρ-επ(ανοίκτης)
ἱερ-ών(υμος), ἱερ-ων(ύμου)
ἱμαντ-ελ(ίκτης)
ἱππ-αγ(ρέτης)
ἱππ-αλ(εκτρύων)
ἵππ-αρ(χος), ἱππ-άρ(χης), ἱππ-αρ(μοστὴς)
ἱππ-ερ(αστὴς)
ἱππ-ήλ(ατος), ἱππ-ηλ(άτης)
ἱππ-ημ(ολγὸς)
ἵππ-ου(ριος), ἱππ-ού(ριδος)
ἰσ-ά(νεμος), ἰσ-α(νέμου)
ἰσ-ή(λικος), ἰσ-ῆ(λιξ), ἰσ-η(μερία)
ἰσ-όνε(ιρος), ἰσ-ονε(ίρου)
ἴσχ-αι(μος), ἰσχ-αί(μου)
καθ-
κακ-άγ(γελος), κακ-αγ(γέλου)
κακ-αν(δρία)
καρ-ρέ(ζουσα), καρ-ρε(ζούσης)
κατ-αγγ(έλω)
κατ-αγί(ζω), κατ-αγι(νέω)
κατ-άγν(υμι), κατ-αγν(ύω)
♦ κατα-γνυπ(όω)
κατ-αγο(ρεύω)
κατ-άγω, κατ-άγε(ις), κατ-άγο(μεν)
κατ-αγω(γὴ)
κατ-ᾴ(δω)
κατ-αθυμ(έω)
♦ κατα-θυμί(ου)
κατ-αι
κατ-ακον(τίζω)
κατ-ακού(ω)
κατ-άκρας
κατ-αλείφ(ω)
κατ-αλέω, κατ-αλέε(ις), κατ-αλέο(μεν)
κατ-αλλ(άσσω)
κατ-αλοά(ω)
κατ-αμά(ομαι), κατ-αμώ(μεθα), κατ-αμῶ(μαι), κατ-αμε(ῖσαι)
κατ-αμελ(έω)
κατ-αμπ(έχω)
κατ-αμύν(ομαι), κατ-αμυν(όμεθα)
κατ-αμύσ(σω), κατ-αμυσ(σόμεθα)
κατ-αμύτ(τω), κατ-αμυτ(τόμεθα)
κατ-αναλ(ίσκω)
κατ-ανασκ(ύλλω)
κατ-ανθ(ρακόω)
κατ-άν, κατ-αν
♦ κατα-νω(τίζομαι)
κατ-άξι(ος), κατ-αξί(ου)
κατ-απειλ(έω)

κατ-άπτομ(αι)
κατ-άρ, κατ-αρ
♦ κατα-ριγ(ηλὸς)
♦ **κατά-ρρ, κατα-ρρ**
κατ-ασβ(έννυμι)
κατ-ασθ(μαίνω)
κατ-ασκέ(ω), κατ-ασκῶ, κατ-ασκεῖ(ς), κατ-ασκο(ῦν)
κατ-ασπάζ(ομαι)
κατ-αστερ(ίζω)
κατ-αυ
κατ-εγγ(υάω)
κατ-έδ(ω)
♦ κατέ-δρ(αθον)
κατ-εἰ(βω), κατ-ει(κάζω)
κατ-εῖ(πα)
♦ κα-τεῖδ(ον), κα-τεῖναι
κατ-ελ(αύνω)
κατ-εμ(έω)
κατ-εν(αίρω)
κατ-εξ(ανίσταμαι)
κατ-επ(αγγέλλομαι)
κατ-ερ(γάζομαι)
κατ-εσ(θίω)
κατ-ευ(ημερεύω)
κατ-εφ(άλλομαι)
κατ-έχ(ω)
κατ-ή, κατ-η
κατ-ί, κατ-ι
κάτ-ο, κατ-ό, κατ-ο
κατ-ύ, κατ-υ
κατ-ωμ(άδιος)
κατ-ωχ(ριάω)
καχ-ε(ξία)
κεν-α(γὴς)
κεντρ-ην(εκὴς)
κέρκ-ου(ρος)
κερ-οἰ(αξ), κερ-οι(άκων)
κεφαλ-αλ(γὸς)
κεφαλ-ηγ(ερέτης)
κλασ-αυ(χενεύομαι)
κοιλ-ωπ(ὸς)
κορυθ-ά(ἴξ), κορυθ-α(ἴκων)
κυαν-ώ(πιδος), κυαν-ῶ(πις)
κύκλ-ωψ, κύκλ-ωπ(ος), κυκλ-ώπ(ων)
κυν-αλ(ώπηξ)
κυν-όδ(ους), κυν-οδ(όντων)
κυν-ώπ(ης), κυν-ῶπ(ις), κυν-ωπ(ίδων)
κωλ-α(κρέτης)
κωμ-άρ(χης), κωμ-αρ(χίου)
κωμ-ω(δὸς)
κωμ-ηλ(άτης)
κωπ-ήρ(ης)
λαβρ-α(γόρης)
λευκ-ανθ(ὴς)
λεύκ-ασ(πις), λευκ-άσ(πιδος), λευκ-ασ(πίδων)
λευκ-ή(ρετμος), λευκ-η(ρέτμου)
λεύκ-ι(ππος), λευκ-ἱ(ππου)
λευκ-ώ(λενος), λευκ-ω(λένου)
λευχ-ε(ιμονέω)
λοχ-αγ(ὸς)
λυκ-αυ(γὴς)
λυκ-ου(ρία)
λύσ-αν(δρος), λυσ-άν(δρου)
μακρ-αἰ(ων), μακρ-αι(ώνων)
μακρ-η(γορέω)
μεγάλ-α(υχος), μεγαλ-ά(δικος), μεγαλ-α(δίκου)
μεγαλ-ή(τωρ), μεγαλ-η(γόρος)
μεγαλ-ώ(νυμος), μεγαλ-ω(νύμου)
μεγ-αυ(χὴς)
μέθ-, μεθ-
♦ μέ-θυ, με-θύ(ω), με-θυ(μναῖος)
♦ μεθ-ύστ(ερος), μεθ-υστ(έρου)
μελάν-α(ιγις), μελαν-α(ίγιδος)
μελαν-ε(ίμων)
μελαν-ύ(δρος)
μελ-ω(δὸς)
μεσ-άγ(κυλον), μεσ-αγ(κύλου)
μέσ-ακ(τος), μεσ-άκ(του)
μέσ-αυ(λος), μεσ-αύ(λου)
μεσ-εγ(γυάω)
μεσ-ημ(βρία)
μεσ-ή(ρης), μεσ-ῆ(ρες)
μεσ-ἰδ(ιος), μεσ-ιδ(ίου)
μεσό-δμ(η), μεσο-δμ(ῶν)
μέσσ-αυ(λος), μεσσ-αύ(λου)
μεσσ-ή(ρης), μεσσ-ῆ(ρες)
μετ-άγγ(ελος), μετ-αγγ(έλου)
μετ-άγω, μετ-άγε(ις), μετ-άγο(μεν)
μετ-αἰ(ρω), μετ-αἰ(σσω), μετ-αι(τέω)
μετ-αλλά(σσω)
μετ-αμπ(έχω)
μετ-αμύ(νω)
μετ-αμφ(ιάζω)
μετ-αμώ(νιος), μετ-αμω(νίου)
μετ-ανίστ(ημι)
μετ-αῦ(θις), μετ-αυ(δάω)
μέτ-ε, μετ-έ, μετ-ε
μετ-ἱ(στημι)
μέτ-ο, μετ-ό, μετ-ο
μέτ-ω, μετ-ώ, μετ-ω
μηδ-α(μὰ)
μηδ-έτ(ερος), μηδ-ετ(έρου)
μηθ-(εἰς)
μήλ-ωψ, μήλ-οπ(ος), μηλ-όπ(ων)
μητρ-α(γύρτης)
μικρ-α(δικητὴς)
μιμ-ω(δὸς)
μιξ-α(ρχαγέτας)
μισ-ά(δελφος), μισ-α(δέλφου)
μισγ-ά(γγεια), μισγ-α(γγείας)
μισ-έλ(λην), μισ-ελ(λήνων)
μογισ-αφ-ε(δάφα)
μοιχ-άγ(ρια), μοιχ-αγ(ρίας)
μον-άμ(πυξ), μον-αμ(πύκων)
μόν-αρ(χος), μον-άρ(χου), μον-αρ(χικὸς)
μον-αυ(λέω)
μον-ήμ(ερος), μον-ημ(έρου)
μον-ήρ(ης), μον-ῆρ(ες)

μόν-ιπ(πος), μον-ίπ(που)
μον-όδ(ους), μον-οδ(όντων)
μον-ω(δέω)
μον-ώψ, μον-ὠψ, μον-ῶπ(ος), μον-ώπ(ων)
μυθ-ίαμ(βος), μυθ-ιάμ(βου)
μυρ-αλ(οιφία)
μυρ-ε(ψὸς)
μυστ-αγ(ωγέω)
νεκρ-άγ(γελτος), νεκρ-αγ(γέλτου)
νεφελ-ηγ(ερέτης)
νεκρ-ακ(αδημία)
νουν-εχ(όντως)
νυκτ-εγ(ερτέω)
νυκτ-ηγ(ορέω)
νυκτ-ηρ(εφὴς)
νυκτ-ού(ρων), νυκτ-οῦ(ρος)
ξεν-αγ(ωγὸς)
ξεν-απ(άτης)
ξεν-ηλ(ατέω)
ὀδ-η(γὸς)
ὀθ-οὔ(νεκα), ὀθ-ού(νεκα)
οἰκ-ουρ(ὸς)
οἰκ-ώφ(ελία)
οἰν-άνθ(η), οἰν-ανθ(ῶν)
οἰν-ερ(αστὴς)
οἰν-οφ, οἴν-οπ(ος), οἰν-όπ(ων)
οἷον-εί, οἰον-εἰ
οἰόσ-περ, οἰόσ-τε
ὀκτακισ-(χίλιοι)
ὀκτ-ήρ(ης), ὀκτ-ῆρ(ες)
ὀκτωκαιδεκ-έ(της)
ὀλιγ-αν(δρία)
ὀλιγ-άρ(χης), ὀλιγ-αρ(ιστία)
ὀλιγ-ότε(ρος), ὀλιγ-οτέ(ρου)
ὀλίγ-ωρ(ος), ὀλιγ-ώρ(ου)
ὅμ-αι(μος), ὁμ-αί(μου)
ὁμ-άλ(ικος), ὁμ-ᾶλ(ιξ)
ὅμ-αυ(λος), ὁμ-αύ(λου)
ὅμ-ευ(νος), ὁμ-εύ(νου), ὁμ-ευ(νέτης)
ὅμ-η(ρος), ὁμ-ή(ρου), ὁμ-η(γερὴς)
ὅμ-ι(λος), ὁμ-ί(λου), ὁμ-ι(λία)
ὅμ-ο(ρος), ὁμ-όρ(ου), ὁμ-ορ(όφιος)
ὅμ-ουρ(ος), ὁμ-ούρ(ου)
ὁμ-ών(υμος), ὁμ-ων(ύμου)
ὁμ-ώρ(οφος), ὁμ-ωρ(όφιος)
ὀν-ηλ(άτης)
ὀξ-άλ(μη)
ὁπωσ-οῦν, ὁπωσ-τι(οῦν)
ὁρκ-ωμό(σιον), ὁρκ-ωμο(σίου)
ὁσ-ημ(έραι)
ὅσ-τε
ὅσ-τις
ὅτ-αν
οὐδ-οπωσ-τι(οῦν), οὐδ-οσ(τισοῦν)
οὐθ-αμ(ῶς)
οὔκ-(ουν), οὐκ-(οῦν)
♦ οὔ-κω(ν)
ὀφ-ώ(νιον), ὀφ-ω(νίου)
παιδ-ολ(έτειρα)
παλαί-χθ(ων), παλαι-χθ(όνων)
παλίν-, παλιν-
παλ-ίω(ξις), παλ-ιώ(ξεως)
πάν-, παν-
♦ πα-νὸς, πα-νός, πα-νὶ, πα-νί, πά-να
♦ πά-νυ
παρ-άγ(ω), παρ-αγ(γέλλω)
♦ παρα-γε(ύω)
♦ παρα-γη(ράω)
♦ παρα-γί(γνομαι), παρα-γι(γνόμεθα)
♦ παρά-γρ(αμμα), παρα-γρ(αφὴ)
♦ παρα-γυ(μνόω)
παρ-αε(ίδω)
παρ-αί(σιος), παρ-αϊ(σσω), παρ-αι(νέω), παρ-αϊ(σσόμεθα)
παρ-ακμ(άζω)
παρ-ακολ(ουθέω)
παρ-ακον(άω)
παρ-άκο(υσμα), παρ-ακο(ύω)
πάρ-αλ(ος), παρ-άλ(ου), παρ-αλ(ία)
♦ παρα-λά(μπω), παρα-λα(μβάνω)
♦ παρα-λέγ(ω)
♦ παρά-λει(ψις), παρα-λεί(πω), παρα-λει(πτέον)
♦ παρ-αλείφ(ω)
♦ παρά-λιμ(νος), παρα-λίμ(νου)
♦ παρά-λογ(ος), παρα-λόγ(ου), παρα-λογ(ίζομαι)
♦ παρά-λυ(σις), παρα-λύ(σεως), παρα-λυ(πέω)
παρ-αμα(ρτάνω)
παρ-αμβ(λύνω)
παρ-αμε(ίβω)
♦ παρα-μέν(ω)
♦ παρα-μετ(ρέω)
παρ-αμπ(έχω)
παρ-ανα(γιγνώσκω)
♦ παρα-ναι(ετάω)
παρ-ανί(ημι), παρ-ανι(σχόμεθα)
♦ παρα-νικ(άω)
παρ-ανοίγ(ω)
πάρ-αντα
παρ-άο(ρος), παρ-αό(ρου)
παρ-απατ(άω)
παρ-απαφ(ίσκω)
παρ-άπτ(ω)
παρ-άρ(θρησις), παρ-αρ(θρήσεως)
♦ παρά-ρρ(υμα), παρα-ρρ(ύματος)
πάρ-αυ(λος), παρ-αύ(λου), παρ-αυ(δάω)
πάρ-ε(δρος), παρ-έ(δρου), παρ-ε(γγράπτου)
♦ πα-ρέω, πα-ρέε(ις)
πάρ-η(μαι), παρ-ή(γορος), παρ-ῆ(λιξ), παρ-η(βάω)
πάρ-ι(σος), παρ-ί(σου), παρ-ι(ππεύω)
πάρ-ο(δος), παρ-ό(δου), παρ-ο(δεύω)
♦ πά-ρος
♦ πα-ρόν(τος)
παρ-ρη(σία)
παρ-ύ(φαινον), παρ-υ(φαίνω)
πάρ-ω(χρος), παρ-ώ(χρου), παρ-ω(νυμία)
♦ πα-ρών, πα-ρὼν
πατρ-άγ(αθος), πατρ-αγ(αθία)
πατρ-αλ(οίας)
πατρ-ωνύ(μιος), πατρ-ωνυ(μίου)

παυσ-άνε(μος), παυσ-ανέ(μου)
πεδ-αί(χμιος), πεδ-αι(χμίου)
πεδ-άο(ρος), πεδ-αό(ρου)
πεδ-άρ(σιος), πεδ-αρ(σίου)
πέζ-αρ(χος), πεζ-άρ(χου)
πεζ-έτ(αιρος), πεζ-ετ(αίρου)
πειθ-ά(νωρ), πειθ-α(νάγκη)
πειθ-ή(νιος), πειθ-η(νίου)
πέλ-οψ, πελ-όπ(ειος), πελ-οπ(όννησος)
πεμπάδ-α(ρχος), πεμπαδ-ά(ρχου)
πεμπ-ώ(βολον), πεμπ-ω(βόλου)
πενθ-ήμε(ρος), πενθ-ημέ(ρου), πενθ-ημε(ρία)
πενθ-ημι(ποδιαῖος)
πέντ-αθ(λον), πεντ-άθ(λου)
πεντ-ήρ(ης), πεντ-ῆρ(ες)
πλεον-έ(κτημα), πλεον-ε(κτέω)
πλήξ-ιπ(πος), πληξ-ίπ(που)
πλησ-ίσ(τιος), πλησ-ισ(τίου)
πλουθ-υ(γίεια)
πλούτ-αρ(χος), πλουτ-άρ(χου)
ποδ-αβ(ρὸς)
ποδ-άγ(ρα), ποδ-αγ(ρῶ)
ποδ-έν(δυτος), ποδ-εν(δύτου)
ποδ-ήρ(ης), ποδ-ῆρ(ες), ποδ-ηγ(ὸς)
ποιμ-ά(νωρ), ποιμ-α(νώρων)
ποσσ-ή(μαρος), ποσσ-ῆ(μαρ), ποσσ-η(μάρων)

πρόσ-, προσ-

- προ-σταυ(ρόω)
- προ-στέλλ(ω)
- προ-στέν(ω), προ-στεν(άζω)
- πρό-στερν(ος), προ-στέρν(ου), προ-στερν(ίδιον)
- πρό-στῳ(ον), προ-στῴ(ου), προ-στῷ(ον)
- προ-συ(γγίγνομαι)
  - προσ-υπ(ερβάλλω)
- πρό-σφαγ(μα), προ-σφάγ(ματος)

πρωθ-ή(βη)
πρωτ-αγ(ωνιστὴς)
πυγ-άρ(γος)
πυθ-αγ(όρας)
πυλ-αγ(όρος)
πυλ-άρ(της), πυλ-αρ(τῶν)
πυλ-ωρ(ὸς)
πυρ-άγ(ρα), πυρ-αγ(ρῶν)
πυρ-ακ(τέω)
ῥαψ-ῳ(δὸς)
ῥιζ-ωρ(υχέω)
ῥίψ-ασ(πις), ῥιψ-άσ(πιδος)
ῥίψ-οπ(λος), ῥιψ-όπ(λου)
σιτ-αγ(ωγὸς)
σκληρ-α(γωγέω)
σκυθρ-ωπ(ὸς)
σπεύσ-ιπ(πος), σπευσ-ίπ(που)
σπονδ-αρ(χία)
στέγ-αρ(χος), στεγ-άρ(χου)
στεροπ-ηγ(ερέτης)
στρατ-ηγ(ὸς)
στρατ-ηλ(άτης)

σύν-, συν-

σωμ-ασ(κέω)
τέθρ-ιπ(πος), τεθρ-ίπ(που), τεθρ-ιπ(ποβάτης)
τειχ-ή(ρης), τειχ-ῆ(ρες)
τεκν-ολ(έτειρα)
τερψί-μ(βροτος), τερψι-μ(βρότου)
τεσσαρεσ-(καίδεκα), τεσσερεσ-(καίδεκα)
τετρ-άρχ(ης), τετρ-αρχ(ῶν)
τετρ-ήρ(ης), τετρ-ῆρ(ες)
τετρ-ώρο(φος), τετρ-ωρό(φου)
τηλ-ουρ(ὸς)
τηλ-ωπ(ὸς)
την-άλ(λως)
τιμ-αλφ(ὴς)
τοιχ-ωρ(ύχος)
τόξ-αρ(χος), τοξ-άρ(χου)
τοξ-ή(ρης), τοξ-ῆ(ρες)
τραγ-έλ(αφος), τραγ-ελ(άφου)
τραγ-ῳ(δὸς)
τραπ-έμπ(αλιν)
τρεις-καί(δεκα)
τρισ-άθ(λιος), τρισ-αθ(λίου)
τρισ-άρ(ιθμος), τρισ-αρ(ίθμου)
τρισ-άσ(μενος), τρισ-ασ(μένου)
τρισ-έ(ωλος), τρισ-ε(υδαίμων)
τρισ-μα(κάριος)
τρισ-μύ(ριοι), τρισ-μυ(ρίων)
τρισ-χ(ίλιοι)
τροχ-ήλ(ατος), τροχ-ηλ(άτης)
τυνβ-α(ύλης), τυμβ-ή(ρης), τυμβ-ῆ(ρες), τυμβ-ω(ρύχος)
ὑμν-ῳ(δία)
ὕπ-α(ιθρος), ὑπ-ά(γω), ὑπ-α(γορεύω)

- ὕ-παιθα, ὑ-παί, ὑ-παὶ
- ὕ-παρ

ὕπ-ε(ιμι), ὑπ-έ(γγυος), ὑπ-ε(γείρω)

- ὑ-πείρ, ὑ-πεὶρ

ὕπερ-, ὑπέρ-, ὑπερ-

- ὑπ-ερεθ(ίζω)
- ὑπ-ερεί(δω)
- ὑπερή-φα(νος), ὑπερη-φά(νου), ὑπερη-φα(νία)
- ὑπ-έρυθ(ρος), ὑπ-ερύθ(ρου), ὑπ-ερυθ(ριῶ)
- ὑπ-έρχ(ομαι), ὑπ-ερχ(όμην)

ὑπ-ημ(ύω)
ὑπ-ήν(εμος), ὑπ-ην(έμιος)
ὑπ-ηο(ῖος)
ὑπ-ηρ(έτης)
ὑπ-ηχ(έω)
ὑπ-ί(ημι), ὑπ-ι(σχνέω)
ὑπ-οι(δέω)
ὑπ-ού(λως), ὑπ-ου(δαῖος)

- ὑπούρ-γ(ημα), ὑπουρ-γ(έω)

ὑπ-ώπ(ιον), ὑπ-ωπ(ίου)
ὑπ-ώρ(εα), ὑπ-ωρ(όφιος)

ὕφ-, ὑφ-

- ὑ-φορβ(ὸς)

ὑψ-α(γόρας)
ὑψ-ε(ρεφὴς)
ὑψ-η(γόρος)
ὑψ-όρ(οφος), ὑψ-ορ(όφου)
φερ-έγ(γυος), φερ-εγ(γύου)
φερ-ώνυ(μος), φερ-ωνύ(μου)

φθιν-όπ(ωρον), φθιν-οπ(ώρου)
φθισ-ή(νωρ), φθισ-η(νώρων)
φίλ-, φιλ-
♦ φι-λέω, φι-λῶ(μεν), φιλέε(ις), φιλεῖ(ς), φι-λέο(μεν)
φί-λι(ος), φι-λί(α), φι-λι(κὸς)
♦ φιλ-ιατ(ρέω)
♦ φίλ-ιππ(ος), φιλ-ίππ(ου), φιλ-ιππ(ικὸς)
φρεν-ήρ(ης), φρεν-ῆρ(ες)
φρούρ-αρ(χος), φρουρ-άρ(χου)
φυγ-αίχ(μης)
φύλ-αρ(χος), φυλ-άρ(χου)
φύλ-οπ(ις), φυλ-όπ(ιδος), φυλ-οπ(ίδων)
φωτ-αγ(ωγὸς)
χαλ-αργ(ὸς)
χαλιν-αγ(ωγέω)
χάλκ-ασ(πις), χαλκ-άσ(πιδος), χαλκ-ασ(πίδων)
χαλκ-έμ(βολος), χαλκ-εμ(βόλου)
χαλκ-ήλ(ατος), χαλκ-ηλ(άτου)
χαλκ-ήρ(ης), χαλκ-ῆρ(ες)
χαμ-εύ(νιον), χαμ-ευ(νίου)
χειρ-αγ(ωγὸς)
χειρ-απ(τάζω)
χειρ-αψ(ία)
χεν-όσ(ιρις)
χην-αλ(ώπηξ)
χορ-α(γὸς)
χορ-η(γὸς)
χρυσ-α(ίετος), χρυσ-ά(μπτυξ)
χρυσ-ελ(εφαντήλεκτρος)
χρυσ-έν(δετος), χρυσ-εν(δέτου)
χρυσ-ερ(αστὴς)
χρυσ-ήλ(ατος), χρυσ-ηλ(άκατος)
χρυσ-ῆρ(ες)
χρυσ-όρο(φος), χρυσ-ορό(φου)
χρυσ-ων(έω)
χρυσ-ώπ(ιδος), χρυσ-ῶπ(ις), χρυσ-ωπ(ίδων)
χρυσ-ωρ(υχέω)
ψευδ-ά(ττικος), ψευδ-α(λέξανδρος)
ψευδ-εν(έδρα)
ψευδ-επ(ίγραφος)
ψευδ-ηγ(ορέω)
ψευδ-ηρ(ακλῆς)
ψευδ-όρ(χιος), ψευδ-ορ(χίου)
ψυχρ-ήλ(ατος), ψυχρ-ηλ(άτου)
ὠμ-ησ(τὴς)
ὡσ-αν-εί, ὡσ-αν-εὶ
ὡσ-αύτως
ὡσ-εί, ὡσ-εὶ
ὡσπερ-εί, ὡσπερ-εὶ

6 The fundamental rules of Latin hyphenation, and the corresponding patterns

Taken from [Chi] (9.56–9.59), here are the rules of Latin hyphenation and the necessary patterns ($v_n, n \geq 1$ will be vowels, including æ and œ, and $c_n, n \geq 1$ consonants):

1. A LATIN WORD HAS AS MANY SYLLABLES AS IT HAS VOWELS OR DIPHTHONGS (æ, au, ei, eu, œ, ui). Concerning word division, this rule should be interpreted as "vowel clusters should be separated, except when they form a diphthong": v_1–v_2, for $(v_1, v_2) \notin \{(a,e),(a,u),(e,i),(e,u),(o,e),(u,i)\}$. The necessary patterns are `a1a`, `a1æ` (since TEX considers æ as one character) `a1i`, `a1o`, `a1œ` (same remark) `e1a`, `e1æ`, `e1e`, `e1o`, `e1œ`, `i1a`, `i1æ`, `i1e`, `i1o`, `i1œ`, `i1i`, `i1u`, `o1a`, `o1æ`, `o1i`, `o1o`, `o1œ`, `o1u`, `u1a`, `u1æ`, `u1e`, `u1o`, `u1œ`, `u1u`.
2. WHEN A SINGLE CONSONANT OCCURS BETWEEN TWO VOWELS, DIVIDE BEFORE THE CONSONANT: v_1–c_1v_2. The required patterns will be `1ba`, `1bæ` ... `1zu`.
3. IN THE CASE OF TWO OR MORE CONSONANTS, DIVIDE BEFORE THE LAST CONSONANT EXCEPT IN THE COMBINATIONS: MUTE (*p*, *ph*, *b*, *t*, *th*, *d*, *c*, *ch*, *g*) + LIQUID (*l*, *r*), AND *qu* OR *gu*. The first part of this rule can be expressed as
   - $v_1c_1 \ldots c_{n-3}$–$c_{n-2}c_{n-1}c_nv_2$ if $(c_{n-2}, c_{n-1}) \in \{(p,h),(t,h),(c,h)$ and $c_n \in \{l,r\}$,
   - $v_1c_1 \ldots c_{n-2}$–$c_{n-1}c_nv_n$ if $c_{n-1} \in \{p,b,t,d,c,g\}$ and $c_n \in \{l,r\}$, and
   - $v_1c_1 \ldots c_{n-1}$–c_nv_2, otherwise.

   The required patterns will be `ph2l`, `ph2r` ... `ch2r`, `p2l` ... `g2r` and `p2h`, `t2h`, `c2h` which is not stated in the rule but seems to be an implicit consequence, especially since "th", "ph", "ch" are just transliterations of the Greek letters θ, φ, χ.

   The second part of the rule ("and *qu* or *gu*") is not clear. According to the examples given in [Chi] ("e-quus", "lin-gua") the author estimates that the correct interpretation is "*consider 'qu' and 'gu' as single consonants*". It follows from this interpretation, that these two clusters form exceptions to rule 1 and that hyphenations "qu-a", "gu-a" ... "gu-u" should be prohibited. The required patterns are `qu2a` ... `gu2u`.
4. COMPOUND WORDS ARE SEPARATED FIRST INTO THEIR COMPONENT ELEMENTS; WITHIN EACH ELEMENT THE FOREGOING RULES APPLY. The patterns needed to fulfill this rule will be discussed in next section.

There is no mention of minimal left and right hyphenations; after comparing several Latin editions, the author considers the traditional values for `\lefthyphenmin` and `\righthyphenmin` to be 2 and 3 (as in English).

7 Hyphenation of Latin compound words

As with ancient Greek, the first idea would be to make patterns out of *roots*: "ab" is such a root, a pattern `ab` would insure hyphenation of "ab-undare", "in-ab-undare" and so forth. The problem is again that the cluster "ab" is contained in thousands of other words, where it should be hyphenated as "a-b" ("la-bor", "pro-ba-bi-lis", "fa-bu-la", etc.). Therefore the same method is applied as in ancient Greek: *only beginnings of words are taken into account.* As the reader will see in next section where all patterns are listed, hyphenation **ab-** at the beginning of a word is a general rule, with certain exceptions ("a-bacus", "a-bitus", and so forth). To hyphenate correctly the word "abs-cidere", a pattern "abs-ci" has been introduced; this pattern produces wrong hyphenation of word "ab-scindere", and so a second pattern "ab-scin" is introduced.

This explains the format of the pattern list in next section: indentation and the symbol ♦ indicate entries, exceptions and exceptions to exceptions. The endings of words placed between parentheses are just examples, THEY ARE NOT TAKEN INTO ACCOUNT IN PATTERN CONSTRUCTION. Entries in boldface are "general rules". For these, no example is given. The same format has been used in the list of patterns for ancient Greek compound words, in section 5.

8 Patterns for Latin compound words

ab-
- ♦ a-bac*(us)*
- ♦ a-bit*(us)*
- ♦ abs-ce*(dere)*
- ♦ abs-ci*(dere)*
  - ♦ ab-scin*(dere)*
- ♦ abs-co*(ndere)*
- ♦ abs-que
- ♦ abs-te*(mius)*
- ♦ abs-ti*(nere)*
- ♦ abs-tr*(ahere)*

ad-
a-gnasc*(i)*
amb-i*(gere)*
antid-ea
a-sc*(endere)*
a-sperg*(ere)*
a-spern*(ari)*
a-spi*(cere)*
a-ste*(rnere)*
a-sti*(pulari)*
a-stre*(pere)*
a-strin*(gere)*
a-strue*(re)*
a-stup*(ere)*
cav-ȷ*(dium)*
circum-
cis-al*(pinus)*
cis-rh*(enanus)*
com-
- ♦ co-met*(es)*
- ♦ co-mi*(cus)*
- ♦ co-mff*(dia)*

de-sc*(endere)*
de-sp*(icere)*
de-st*(inare)*
di-ch*(oreus)*
di-gn*(oscere)*
dir-im*(ere)*
di-scrib*(ere)*
di-sperg*(ere)*
di-spi*(cere)*
di-sta*(re)*
di-sting*(uere)*
ex-
id-eo
in-
- ♦ i-nan*(is)*
  - ♦ ianim*(us)*
- ♦ ind-ue*(re)*
- ♦ ind-ep*(tus)*
- ♦ ind-ig*(es)*
- ♦ ind-ip*(iscor)*
- ♦ ind-ue*(re)*
- ♦ ini-ti*(a)*

inter-
long-ȷv*(us)*
neg-oti*(um)*
ob-
- ♦ obli-vi*(o)*
- ♦ oblon-*(gus)*
- ♦ obff-di*(re)*

pȷn-ins*(ula)*
per-, post-, prȷ-
prod-ir*(e)*
prod-es*(se)*
prod-ig*(ere)*
pro-sc*(ȷnium)*
pro-sp*(ectus)*
pro-st*(are)*
quinc-un*(x)*
quot-an*(nis)*
re-
- ♦ red-*(arguere)*
  - ♦ re-don*(are)*
  - ♦ re-dor*(mire)*

♦ re-duc*(ere)*
sat-ag*(ere)*
satis-ac*(cipere)*
sem-un*(ciarius)*
sem-us*(tulatus)*
sesc-en*(naris)*
sic-ut*(i)*
sub-, super-
su-scr*(ibere)*
su-sp*(icere)*
ter-un*(cius)*
trans-ab-*(ire)*
trans-
♦ tran-sil*(ire)*
♦ tran-su*(ere)*
♦ trans-us*(que)*
vel-ut*(i)*

9 Examples

Follow some examples of hyphenated ancient Greek and Latin texts. The symbol ÷ indicates hyphenation using patterns from sections 5 and 8.

Ἐντεῦ÷θεν ἐξ÷ε-λαύ-νει σταθ-μοὺς τρεῖς πα-ρα-σάγ-γας πεν-τε-καί-δε-κα ἐ-πὶ τὸν Εὐ-φρά-την πο-ταμόν, ὄν-τα τὸ εὖ-ρος τετ-τά-ρων στα-δί-ων· καὶ πό-λις αὐ-τό-θι ᾧ-κεῖ-το με-γά-λη καὶ εὐ-δαί-μων Θά-ψα-κος ὄ-νο-μα. Ἐνταῦ÷θα ἔ-μει-νεν ἡ-μέ-ρας πέν-τε. Καὶ Κῦ-ρος με-τα-πεμ-ψά-με-νος τοὺς στρατ÷η-γοὺς τῶν Ἑλ-λή-νων ἔ-λε-γεν ὅ-τι ἡ ὁ-δὸς ἔ-σοι-το πρὸς βα-σι-λέ-α μέ-γαν εἰς Βα-βυ-λῶ-να· καὶ κε-λεύ-ει αὐ-τοὺς λέ-γειν ταῦ-τα τοῖς στρα-τι-ώ-ταις καὶ ἀ-να-πεί-θειν ἔ-πε-σθαι. Οἳ δὲ ποι-ή-σαν-τες ἐκ-κλη-σί-αν ἀπ÷ήγ-γε-λον ταῦ-τα· οἱ δὲ στρα-τι-ῶ-ται ἐ-χα-λέ-παι-νον τοῖς στρατ÷η-γοῖς, καὶ ἔ-φα-σαν αὐ-τοὺς πά-λαι ταῦτ' εἰ-δό-τας κρύ-πτειν, καὶ οὐκ ἔ-φα-σαν ἰ-έ-ναι, ἐ-ὰν μή τις αὐ-τοῖς χρή-μα-τα δι-δῷ, ὥ-σπερ τοῖς προ-τέ-ροις με-τὰ Κύ-ρου ἀ-να-βᾶ-σι [πα-ρὰ τὸν πα-τέ-ρα τοῦ Κύ-ρου], καὶ ταῦ-τα οὐκ ἐ-πὶ μά-χην ἰ-όν-των, ἀλ-λὰ κα-λοῦν-τος τοῦ πα-τρὸς Κῦ-ρον. Ταῦ-τα οἱ στρατ÷ηγοὶ Κύ-ρῳ ἀπ÷ήγ-γελ-λον. Ὁ δ' ὑπ÷έ-σχε-το ἀν-δρὶ ἑ-κά-στῳ δώ-σειν πέν-τε ἀρ-γυ-ρί-ου μνᾶς, ἐ-πὰν εἰς Βα-βυ-λῶ-να ἥ-κω-σι, καὶ τὸν μι-σθὸν ἐν-τε-λῆ μέ-χρι ἄν κα-τα-στή-σῃ τοὺς Ἕλ-λη-νας εἰς Ἰ-ω-νί-αν πά-λιν. Τὸ μὲν δὴ πο-λὺ τοῦ Ἑλ-λη-νι-κοῦ οὕ-τως ἐ-πεί-σθη.

from Ξε-νο-φῶν-τος Κύ-ρου Ἀ-νά-βα-σις. [Ξεν]

Flu-men est Arar, quod per fi-nes Hæ-du-o-rum et Se-qua-no-rum in Rho-da-num in-fluit, in-cre-dibili le-ni-tate, ita ut ocu-lis in ut-ram par-tem fluat iu-di-cari non pos-sit. Id He-lu-e-tii ra-ti-bus ac lintri-bus iunc-tis trans÷i-bant. Vbi per ex-plo-ra-tores Cæ-sar cer-tior fac-tus est tres iam par-tes copia-rum He-lu-e-tios id flu-men tra-du-xisse, quartam fere par-tem citra flu-men Ara-rim re-li-quam esse, de ter-tia ui-gi-lia cum le-gi-o-ni-bus tri-bus e cast-ris pro-fec-tus ad eam par-tem pe-ru-e-nit quæ non-dum flu-men trans÷i-e-rat. Eos im-pe-di-tos et in÷o-pi-nan-tes ad-gres-sus mag-nam par-tem eorum con-ci-dit: re-li-qui sese fugæ man-da-runt atque in pro-xi-mas si-luas ab-di-de-runt.

from Cæ-sa-ris Com-men-ta-rii de Bello Gal-lico. [Cæ]

10 Availability

The files `AGRhyphen.tex` and `LAThyphen.tex` containing hyphenation patterns for ancient Greek and Latin (as described in this paper) are part of the ScholarTEX package.

The hyphenation patterns for modern Greek mentioned in section 2 (file `GRhyphen.tex`) are in the public domain; they are included in Euro-OzTEX and can also be obtained directly from the author. Readers interested in Greek (ancient or modern) are invited to join the `ELLHNIKA` discussion list, by sending the `SUBSCRIBE ELLHNIKA` <*name*> command to `LISTSERV@DHDURZ1.BITNET`.

References

[Aca] Ι. Καλλιτσουνάκης (εἰσηγητής), *Ὀρθογραφικὸν διάγραμμα τῆς Ἀκαδημίας Ἀθηνῶν*, Πρακτικὰ Ἀκαδημίας Ἀθηνῶν, τ. ιδ΄, 1939.

[Bai] A. Bailly, *Abrégé du dictionnaire grec-français*, Hachette, Paris, 1901.

[B–C] H. Bornecque et F. Cauët, *Dictionnaire latin-français*, Librairie classique Eugène Belin, Paris, 1990.

[Cæ] César, *Guerre des Gaules*, trad. par L.-A. Constans, Les belles lettres, Paris 1984.

[Chi] *The Chicago Manual of Style*, The University of Chicago Press, Chicago and London, 1982.

[DEK] D. E. Knuth, *Computers & Typesetting, A: The TEXbook*, Addison-Wesley, Reading, 1989.

[Xen] Xénophon, *Anabase*, trad. par P. Masqueray, Les belles lettres, Paris 1970.

The Exotic Croatian Glagolitic Alphabet

Darko Žubrinić

Dedicated to the memory of Gordan Lederer (1958–1991)

Just for fun, one day I decided to write my own name in Glagolitic letters. I was very proud of the result: ⰄⰀⰓⰍⰑ ⰆⰖⰁⰓⰉⰐⰉⰛ. Without creating TEX Prof. Donald Knuth probably would have never had opportunity to see his name written as

ⰄⰑⰐⰀⰎⰄ ⰍⰐⰖⰕ.

(The h = Ⱈ was dropped because it is not pronounced.)

The origins of the Croatian Glagolitic alphabet are still mysterious. The only thing we can state for sure is that it has existed in my homeland for more than a thousand years, i.e. since the ninth century. Croats have been living in their homeland since the seventh century and they were the first among Slavs to be Christianized. It used to be generally regarded that the Glagolitic alphabet was created by St. Cyrill, a Greek apostle from Thessaloniki, but now there exist several very different theories about its origins. However, the fact that Croats had already been Christianized when St. Cyrill was born (825), together with the unique multiorthographic tradition of written documents (Glagolitic, Latin, Cyrillic) in medieval Croatia, and above all, more than a thousand years' history of Glagolitic script in Croatia, seem to prove that the origins of Glagolitic script are *authentically Croatian.*

One of the earliest Glagolitic inscriptions we know of in Croatia can be seen on a stone monument found in the church of St. Lucy near the city of Bashka on the island of Krk, dating back to around 1100 AD. It is the oldest known monument written in my native tongue which mentions the name of Croatia (i.e. Hrvatska) and the name of the Croatian king Zvonimir.

Through the Glagolitic alphabet Croats kept in touch with other European cultures of the Middle Ages. For example, in 1347 the famous Czech king Charles IV established a Glagolitic convent near Prague, where Croatian priests were teaching the Glagolitic alphabet. Similarly, the Polish king Wladislav II Yagiell organized (in 1390) a Glagolitic convent near Krakow.

Especially interesting is the story of the old Glagolitic book handwritten on the island of Krk in Croatia, that somehow came from Prague to Reims in France. There, for centuries afterwards, French kings were sworn in by putting their hands on this holy book (it still exists).

In 1248, by the decree of Pope Innocent IV, Croats were allowed to practise Glagolitic liturgy (i.e. early Croatian), using holy books written in Glagolitic instead of Latin or Greek. This decision of the Pope was unique in medieval Europe — Croats were the only nation in Europe who were allowed to use their own language in liturgy instead of Latin.

Even today the Glagolitic liturgy is preserved in some parts of Croatia, with priests still singing in early Croatian language as in St. Cyrill's time (the ninth century!). The Glagolitic alphabet has probably been our most important cultural monument for thirteen centuries of difficult, but rich life in Europe.

As for the name of Croatia, let me mention by the way that the French and German names for tie — cravate and die Krawate — were coined from it. It would take us too far from our purpose to tell in detail this very interesting story.

There are thousands of monuments, pergament letters and books written in the Glagolitic alphabet. One of the most beautiful certainly is "Misal" (or ⰏⰉⰔⰀⰎ), printed in 1483, most probably in the Croatian town of Kosinj, only 37 years after Gutenberg's invention, or only six years after the first printed books appeared in Paris and Venice, or 70 years before the first book was printed in Moscow. Like Gutenberg's Bible, it has many ligatures. Unfortunately, in 1493 there was a penetration of the Turkish Ottoman Empire, which was stopped in Croatia (until the XIXth century!). This did not allow a normal development of printing as in other parts of Europe. Despite very difficult conditions many Glagolitic documents bear witness to surprisingly rich cultural activity in medieval Croatia, especially on the island of Krk and the Istrian penisula.

Glagolitic books for Croatian priests were also printed in Venice, which even had two Glagolitic churches at one time, then in Rome. With the help of Croatian protestants books were printed in Wittenberg and Urach in Germany. One of the founders of protestantism in Europe was the Croatian philosopher Flacius Illiricus. The Glagolitic alphabet was also taught in the city of Dubrovnik. Besides in Croatia, Croatian books and manuscripts written in the Glagolitic alphabet are now kept in Rome, Sankt Petersburg, Berlin, Vienna, Innsbruck, Moscow, Copenhagen, London, Oxford, Constantinople, Paris, Tours, New York, Krakow, Porto, Budapest, Trento, Padova, Sienna, and some other places.

There are a few Glagolitic letters that came from Greek, like ф (f), э (e); the letter ш (sh) came

from Hebrew. You will also find these letters in Cyrillic script, created later by the followers of St. Method in Bulgaria on the basis of the Greek uncial script.

Of course, one can find some similarities with other Cyrillic and roman letters, but the difference is considerable. It is interesting to note that the Glagolitic A = ⰰ is almost the same as the Ethiopian `ha`. I learned that in a very interesting article [1].

The complete font, together with numerical values, looks like this:

ⰰ	A	1	ⱁ	O	80
ⰱ	B	2	ⱂ	P	90
ⰲ	V	3	ⱃ	R	100
ⰳ	G	4	ⱄ	S	200
ⰴ	D	5	ⱅ	T	300
ⰵ	E	6	ⱆ	U	400
ⰶ	Ž	7	ⱇ	F	500
ⰷ	Dz	8	ⱈ	H	600
ⰸ	Z	9	ⱉ	Ot	700
ⰹ	Iže	10	ⱋ	Št, Šć, Ć	800
ⰻ	I	20	ⱌ	C	900
ⰼ	J	30	ⱍ	Č	1000
ⰽ	K	40	ⱎ	Š	2000
ⰾ	L	50	ⱑ	Ja, Je	
ⰿ	M	60	ⱓ	Ju	
ⱀ	N	70			

It was created according to the above mentioned "Misal" from 1483. Note that the letter ⱍ = ch looks rather 'chinese'. From the table we see that some of the symbols had only numerical values, like ⰹ = 10.

The letters were also assigned appropriate numerical values, similarly to the old Greek script. For example the year 1254 could have been written as ·ⱍ·ⱄ·ⰾ·ⰳ·. Numbers from 11 to 19 were written in the reverse order, for instance ⰱⰹ = 12. What do you say about the following arithmetic:

ⰷ + ⰳ = ⰱⰹ, ⱉ + ⱆ = ⱍⱃ?

Among many interesting ligatures let me mention a 'three storey m' = ⰿⰿⰿ, which was used for [ml] (our language is not easy — remember the tongue-twisting island Krk), and 'double i' = ⰻⰻ for [ili], which I like very much. Some of the ligatures are represented on the following list:

ⰰⰿ	am	ⰼⱆⱅⱃ	jutr	ⰿⰶ	mž	ⱁⰾⰼⱆ	olju
ⰱⰾ	bl	ⰽⱁ	ko	ⰿⰾ	ml	ⱁⱅ	ot
ⰱⱁ	bo	ⰾⰻ	li	ⰿⰾⱍ	mlč	ⱂⰾ	pl
ⰱⱃ	br	ⰾⱁ	lo	ⰿⱁ	mo	ⱂⱁⰲⱃ	povr
ⰻⰾ	il	ⰾⱅ	lt	ⰿⱄ	ms	ⱄⱁ	so
ⰻⰾⰻ	ili	ⰾⰲ	lv	ⱀⱁ	no	ⱅⰲⱃ	tvr
ⰻⱅ	it	ⰾⰼⱆ	lju	ⱁⰾ	ol	ⰲⱁⰴ	vod
ⰼⱆⱃ	jur	ⰾⰼⱆⰴ	ljud	ⱁⰾⰻ	oli	ⰸⱃ	zr

Ligatures give a special flavor to Glagolitic manuscripts. As an example, the Glagolitic expression for 'moon' is in some manuscripts given by ⰿⱑⱄⱌ (in Croatian: mjesec).

A few words about the TEX-community in Croatia. It is rather widespread. Many students prepare their graduation works using TEX, while among mathematicians it has become a routine means of creating documents. However, we still lack some basic literature, and we are still not organized as in other countries. We hope this will improve through collaboration with your excellent journal.

I would be pleased to contact anyone wishing to learn more about the Glagolitic alphabet. Let me take the opportunity to illustrate it by greeting my friends and colleagues in the USA who know Croatian:

ⱂⱁⰸⰴⱃⰰⰲ ⱄⰲⰻⰿ ⰳⰾⰰⰳⱁⰾⰼⰰⱎⰻⰿⰰ – ⰶⱆⰱⱃⰰ.

I would like to thank Sonja Šterc (Zagreb) and Barbara Beeton for help during the preparation of this article.

References

[1] Abass Andulem: The road to Ethiopic TEX. *TUGboat*, Vol. 10, No. 3 (November 1989), 352–354.

[2] Donald Knuth: *The METAFONTbook*, Addison Wesley, 1986.

◇ Darko Žubrinić
University of Zagreb
ETF, Avenija Vukovar 39, Zagreb
Croatia
Internet:
darko.zubrinic@etf.uni-zg.ac.mail.yu

Fonts

Postnet codes using METAFONT

John Sauter

Abstract

A reimplementation of Dimitri Vulis' Postnet bar codes.

1 Introduction

I was excited to read in *TUGboat* 12, no. 2, about Dimitri Vulis' work with the Postnet bar codes for envelopes. I was determined to include his work into my letter-writing software until I came to the last line of his article, where he says "The macros are copyrighted, though, and I intend to defend them strenuously against unauthorized commercial use."

I was disappointed. This stricture meant that I could not use Mr. Vulis' work, since I sometimes write letters on behalf of a small retail store near my home. I was determined to find a way to use the Postnet codes in spite of Mr. Vulis' limitation.

2 A Different Approach

To avoid violating Mr. Vulis' copyright on his macros, I decided to take a different approach to the problem of producing Postnet codes. I would implement the bar codes using METAFONT as much as possible, with only as much TEX macros as necessary for support. I started by visiting my local Post Office, so I could obtain the information for constructing the bar codes from the source, to avoid any accusation that I had violated Mr. Vulis' copyright by using the dimension information in his macros.

I was pleased to find that the Post Office has liberalized the rules for Postnet codes since Mr. Vulis did his work. The FIM is no longer necessary, and the Postnet code can be placed immediately above the addressee's name, as follows:

John Sauter
9-801128-09 Elizabeth Drive
Merrimack, NH 03054-4576

Placing the Postnet code here avoids the hassle of figuring out how to get the code to appear in the proper corner of the envelope.

3 The METAFONT Font

The Postnet font file begins with identification and setup information.

```
% Postnet font, for USPS barcodes.
%
mode_setup;
%displaying:=0;
"Postnet digits";
font_identifier "POSTNET";
font_coding_scheme "Digits";
```

As Mr. Vulis describes in his article, Postnet represents digits using long and short bars. The first order of business in the font, therefore, is to define the dimensions of the bars and the spacing between them. This information is taken from the United States Postal Service regulations as described in the Domestic Mail Manual (DMM) issue 39 (June 16, 1991) sections 551 and 552, as summarized and explained in *Bar Code Update*, a document provided to me by my Postmaster.

```
% Primary parameters, as specified by
% the U. S. Postal Service.
bar_width#      := 0.020in#;
   % plus or minus 0.005in
half_bar_height#:= 0.050in#;
   % plus or minus 0.010in
full_bar_height#:= 0.125in#;
   % plus or minus 0.010in
bar_spacing#    := 1/22in#;
   % 20 to 24 bars per inch
```

The tolerances placed on the bar dimensions are great enough that any reasonably modern printer should have no trouble producing acceptable bar codes.

I now define some secondary parameters, so called because they are based on the primary parameters.

```
% Secondary parameters
digit_width#:=5/22in#;
   % width of a digit
digit_height#:=full_bar_height#+0.04in#;
   % leave space above bars
digit_depth#:=0.040in#;
   % min space below bars
```

The font parameters are next. These parameters are very simple, since this is a fixed-width font and all the characters are the same height.

```
font_size digit_height#;
font_slant 0;
font_normal_space digit_width#;
font_normal_stretch 0;
font_normal_shrink 0;
```

```
font_x_height half_bar_height#;
font_quad digit_width#;
font_extra_space 0;
```

It is now time to specify pixel-dimensioned versions of the necessary parameters. These will be used when actually drawing characters.

```
define_pixels (bar_width);
define_pixels (half_bar_height);
define_pixels (full_bar_height);
define_pixels (bar_spacing);
define_pixels (digit_width);
define_pixels (digit_height);
define_pixels (digit_depth);
```

Here is an alternate version of Plain METAFONT's `makebox`, which provides more information when printing proofs. It is based on an example in *The METAFONTbook*, Appendix E.

```
def makebox(text r) =
 for y=0,full_bar_height,
   half_bar_height,digit_height,
   -digit_depth:
  r((0,y),(w,y)); endfor % horizontals
 for x=0 step bar_spacing until w:
  r((x,0),(x,h)); endfor % verticals
 r((w,0),(w,h));
enddef;
```

All of the real work in a bar code font is done by drawing bars. It therefore seems fitting to place the bar-drawing macro next. This macro has two explicit parameters, the bar number and the bar height. It defines two points, `t` at the top and `b` at the bottom of the bar, and uses the current pen to draw it. The macro also depends on `bar_pos` to be the left edge of the bar, and increments this value so that the next invocation of the macro will draw the next bar in the following position.

```
def draw_bar (suffix $)
      (expr bar_height) =
   lft x$t = bar_pos*bar_spacing;
   top y$t = bar_height;
   x$b = x$t;
   bot y$b = 0;
   draw (z$t -- z$b);
   labels ($t, $b);
   bar_pos := bar_pos + 1
enddef;
```

Now, following the example of Computer Modern, I have defined macros to provide the beginning and end of each character. These macros are quite simple because of the simple nature of the font. All of the digits have the same width, height and depth, so the only parameter is the character code of the digit. The `end` macro is for aesthetics.

```
def beginpostnetchar (expr char_code) =
  beginchar (char_code, digit_width#,
      digit_height#, digit_depth#);
    bar_pos := 0;
    pickup stdpen;
enddef;

def endpostnetchar =
  endchar;
enddef;
```

We use only a single pen, with a simple shape: it has no height and is the width of a bar. We will use this pen only for vertical strokes.

```
pen stdpen;
stdpen = penrazor xscaled bar_width;
```

Well, it seems I lied about this font only containing digits. We need an additional full bar at the beginning and end of numbers, and as long as we need a separate full bar we should in fairness also have a half bar. We can use the `draw_bar` macro, but not the others since they assume a complete digit.

```
"full bar";
beginchar ("f", bar_spacing#,
    digit_height#, digit_depth#);
  bar_pos := 0;
  pickup stdpen;
  draw_bar (0, full_bar_height);
endchar;

"half bar";
beginchar ("h", bar_spacing#,
    digit_height#, digit_depth#);
  bar_pos := 0;
  pickup stdpen;
  draw_bar (0, half_bar_height);
endchar;
```

Now that the preliminaries are out of the way we can proceed with the digits themselves. Each digit consists of five bars, two full height and three half height. The pattern for each digit is described in *Bar Code Update*.

```
"Digit Zero";
beginpostnetchar ("0");
  draw_bar (0, full_bar_height);
  draw_bar (1, full_bar_height);
  draw_bar (2, half_bar_height);
  draw_bar (3, half_bar_height);
  draw_bar (4, half_bar_height);
endpostnetchar;
```

```
"Digit One";
beginpostnetchar ("1");
  draw_bar (0, half_bar_height);
  draw_bar (1, half_bar_height);
  draw_bar (2, half_bar_height);
  draw_bar (3, full_bar_height);
  draw_bar (4, full_bar_height);
endpostnetchar;

"Digit Two";
beginpostnetchar ("2");
  draw_bar (0, half_bar_height);
  draw_bar (1, half_bar_height);
  draw_bar (2, full_bar_height);
  draw_bar (3, half_bar_height);
  draw_bar (4, full_bar_height);
endpostnetchar;

"Digit Three";
beginpostnetchar ("3");
  draw_bar (0, half_bar_height);
  draw_bar (1, half_bar_height);
  draw_bar (2, full_bar_height);
  draw_bar (3, full_bar_height);
  draw_bar (4, half_bar_height);
endpostnetchar;

"Digit Four";
beginpostnetchar ("4");
  draw_bar (0, half_bar_height);
  draw_bar (1, full_bar_height);
  draw_bar (2, half_bar_height);
  draw_bar (3, half_bar_height);
  draw_bar (4, full_bar_height);
endpostnetchar;

"Digit Five";
beginpostnetchar ("5");
  draw_bar (0, half_bar_height);
  draw_bar (1, full_bar_height);
  draw_bar (2, half_bar_height);
  draw_bar (3, full_bar_height);
  draw_bar (4, half_bar_height);
endpostnetchar;

"Digit Six";
beginpostnetchar ("6");
  draw_bar (0, half_bar_height);
  draw_bar (1, full_bar_height);
  draw_bar (2, full_bar_height);
  draw_bar (3, half_bar_height);
  draw_bar (4, half_bar_height);
endpostnetchar;

"Digit Seven";
beginpostnetchar ("7");
  draw_bar (0, full_bar_height);
  draw_bar (1, half_bar_height);
  draw_bar (2, half_bar_height);
  draw_bar (3, half_bar_height);
  draw_bar (4, full_bar_height);
endpostnetchar;

"Digit Eight";
beginpostnetchar ("8");
  draw_bar (0, full_bar_height);
  draw_bar (1, half_bar_height);
  draw_bar (2, half_bar_height);
  draw_bar (3, full_bar_height);
  draw_bar (4, half_bar_height);
endpostnetchar;

"Digit Nine";
beginpostnetchar ("9");
  draw_bar (0, full_bar_height);
  draw_bar (1, half_bar_height);
  draw_bar (2, full_bar_height);
  draw_bar (3, half_bar_height);
  draw_bar (4, half_bar_height);
endpostnetchar;
```

In the Postnet code a number is more than a string of digits. To be a proper number a string of digits must have a full height bar before and after it. We can use the new facilities of METAFONT version 2 to provide these additional bars as ligatures.

```
%
% ligature table for Postnet font.
% provide tall bars at the beginning
%  and end of numbers.
%
boundarychar := 32;
beginchar (boundarychar, 0, 0, 0);
  endchar;

ligtable ||:
  "0" =:| "f",
  "1" =:| "f",
  "2" =:| "f",
  "3" =:| "f",
  "4" =:| "f",
  "5" =:| "f",
  "6" =:| "f",
  "7" =:| "f",
  "8" =:| "f",
  "9" =:| "f",
```

```
"0": "1": "2": "3": "4": "5": "6":
  "7": "8": "9": 32 |=: "f";
```

And with that, the font description is complete.

```
bye;
```

4 The TEX macros

The font itself is adequate for simple examples, like the one earlier in this article. However, as explained by Mr. Vulis, each number also ends with a check digit. I considered, very briefly, trying to do the check digit computation as a ligature table in the font. I came to the conclusion that METAFONT is the wrong language for such a computation, since the size of the ligature table gets very large with 11-digit numbers. Therefore, I decided to write the check digit code using TEX macros. Mr. Vulis used a very clever technique, but because of his copyright I had to use a different method. After some hunting I found an example of almost exactly what I needed in *The TEXbook*, Appendix D.

In the following pair of macros, `\postnetcheckdigit` takes as its argument a string of digits followed by a vertical bar. It uses `\getpostnetcheckdigit` to set `\count0` to the sum of the digits, and then arranges for token register `\Postnetchecktoken` to be set to the correct checksum digit for the string.

```
\def\getpostnetcheckdigit#1{\ifx#1\end
  \let\next=\relax
  \else\advance\count0 by #1%
    \let\next=\getpostnetcheckdigit\fi
  \next}
\def\postnetcheckdigit#1|{{\count0=0
  \getpostnetcheckdigit#1\end
  \count1=\count0
  \divide\count1 by 10
  \multiply\count1 by 10
  \advance\count0 by -\count1
  \count1=10
  \advance\count1 by -\count0
  \ifnum \count1>9
    \advance\count1 by -10\fi
  \aftergroup\Postnetchecktoken
  \aftergroup=\aftergroup{%
  \expandafter\aftergroup\number\count1
  \aftergroup}%
  }}
```

The check digit must be combined carefully with the rest of the digits so that the ligatures work correctly, placing a full bar before the first digit and after the check digit. In addition, it is convenient to have a macro which specifies the Postnet digits so they can be printed wherever in the letter the style requires. In some cases the Postnet code will be unknown or inappropriate, and so should not be printed.

To accommodate these needs I have a macro `\Postnetdigits` which accepts the digit string, and `\Postnetline` which is used from my letter formatting macros to set a line of Postnet bar codes.

```
\def\Postnetdigits #1{%
  \Postnettoken={#1}%
  \postnettrue}
\def\Postnetline{\ifpostnet
  \expandafter\postnetcheckdigit
  \the\Postnettoken|%
  \Postnettoken=\expandafter\expandafter
  \expandafter{\expandafter
  \the\expandafter\Postnettoken
  \the\Postnetchecktoken}%
  {\Postnetfont \the\Postnettoken\hfil}%
  \fi}
```

In support of the above macros we must declare the token registers and the condition.

```
\newtoks\Postnettoken
\newtoks\Postnetchecktoken
\newif\ifpostnet
\postnetfalse
```

My letter formatting macros are based on *The TEXbook*, Appendix E. I have placed the Postnet code only on the envelope, so only macro `\makelabel` needs modification. The Postnet bar code goes just above `\theaddress`, as follows:

```
\def\makelabel{\endletter\hbox{\vrule
    \vbox{\hrule \kern6truept
     \hbox{\kern6truept
       \vbox to 2truein
       {\hsize=6truein
         \smallheadfont
         \baselineskip9truept
         \returnaddress
         \vfill\vbox {%
           \hskip 2truein\Postnetline}%
         \moveright 2truein
         \copy\theaddress\vfill}%
       \kern6truept}%
     \kern6truept\hrule}%
    \vrule}
  \pageno=0\vfill\eject}
```

5 Conclusion

I thank Mr. Vulis for the motivation his article gave me to re-implement his Postnet bar codes.

This is the first font that I've constructed entirely on my home computer, without support from the mainframes at work. I am using a Commodore Amiga 2000 and AmigaTEX from Radical Eye Software. Tomas Rokicki has provided an excellent implementation of TEX and METAFONT on the Amiga.

I place no restrictions or limitations of any kind on the font and TEX macros in this article. Anyone can use them, or any derivation of them, for any purpose whatsoever. If anyone would like to use the font without the bother of typing in the font file, just get in touch with me at the address below, and I'll give you instructions on how to obtain an electronic copy.

◇ John Sauter
System Eyes Computer Store
Graystone Plaza, 101A
Nashua, NH 03063
603-889-1234

A typewriter font for the Macintosh 8-bit font table

Yannis Haralambous

Macintosh Programmer's Workshop users often write 8-bit code like

```
Replace -c ∞ /•#[∂t ]*define
([¬]+)⊛1 ([¬ ∂t∂n]+)⊛2(≈)⊛3∞/
"∂$SETC ⊛1 := ⊛2∂ ⊛3"
Find •
Replace -c ∞ /for ∂(([¬;]*)⊛1;
([¬<]*)<=([¬;]*)⊛2;[¬∂)]*∂)/∂
"FOR ⊛1 TO ⊛2 DO"
Find •
```

How would you write this code in `verbatim` mode?

A first solution is proposed by Michael Spivak in his LAMS-TEX package: he uses an escape-from-verbatim character, which acts like a backslash. This has the advantage that you can use every possible TEXmacro inside `verbatim`; on the other hand, your `verbatim` code isn't verbatim any more: it needs special treatment (for example, replace all ∂ by "`partial` if " is your escape character, etc.). And moreover, you will have to mix fonts to obtain symbols like ∞, §, etc.

Another elegant solution would be to use virtual fonts to collect the characters from other fonts; but many symbols do not exist in typewriter form.

Here is a "brute force" solution: `MACTT` is a "`CM`-like"[1] font covering the whole Macintosh 8-bit font table. Actually, as you can see in the font table below, the encoding of this font is a mixture of `CM` and Macintosh encoding, in the following sense:

1. The 7-bit part of the table *is* `CMTT`, except for two characters:
   - the ASCII "apostrophe" *'047* is a real straight apostrophe ' instead of the ' in `CMTT`, which in fact is a closing quote.
   - the ASCII "grave" *'140* is a real grave accent `, instead of the `CMTT` ‘, which in fact is an opening quote.
2. On the other hand, the Macintosh font table doesn't use characters *'000* to *'037* and *'177*. In these cases I left the original `CMTT` characters.

Note that even in the original Macintosh font table, characters *'366* and *'367* are similar to characters *'136* and *'176*. Ambiguity is avoided by the fact that all screen-source fonts (like Monaco, Geneva, Chicago) leave positions *'366* and *'367* empty.

Many characters have just been taken from other `CM` fonts (like £, §, •, †, ∫, etc.). Others needed slight changes, like ∞, ¶ (instead of ¶), etc. And finally, some characters have been created from scratch, like ⊛, ™, ∫ and the unavoidable apple .

To use this font, you just have to set `\font\tt=mactt10` at the beginning of your document. As for the METAFONT sources (package `MACTT.SHELL`), they are shareware and can be obtained from all major archives.

[1] To make a "`DC`-like" font out of it, just replace characters *'000* to *'037* and *'177* by the corresponding characters in the `DCTT` font.

Font table of MACTT10

	'0	'1	'2	'3	'4	'5	'6	'7	
'00x	Γ	Δ	Θ	Λ	Ξ	Π	Σ	Υ	"0x
'01x	Φ	Ψ	Ω	↑	↓	'	¡	¿	
'02x	ı	ȷ	`	´	˘	¯	˚	¸	"1x
'03x	ß	ß	æ	œ	ø	Æ	Œ	Ø	
'04x	␣	!	"	#	$	%	&	'	"2x
'05x	(	)	*	+	,	-	.	/	
'06x	0	1	2	3	4	5	6	7	"3x
'07x	8	9	:	;	<	=	>	?	
'10x	@	A	B	C	D	E	F	G	"4x
'11x	H	I	J	K	L	M	N	O	
'12x	P	Q	R	S	T	U	V	W	"5x
'13x	X	Y	Z	[	\	]	^	_	
'14x	`	a	b	c	d	e	f	g	"6x
'15x	h	i	j	k	l	m	n	o	
'16x	p	q	r	s	t	u	v	w	"7x
'17x	x	y	z	{	|	}	~	¨	
'20x	Ä	Å	Ç	É	Ñ	Ö	Ü	á	"8x
'21x	à	â	ä	ã	å	ç	é	è	
'22x	ê	ë	í	ì	î	ï	ñ	ó	"9x
'23x	ò	ô	ö	õ	ú	ù	û	ü	
'24x	†	°	¢	£	§	•	¶	ß	"Ax
'25x	®	©	™	´	¨	≠	Æ	Ø	
'26x	∞	±	≤	≥	¥	µ	∂	Σ	"Bx
'27x	Π	π	∫	ª	º	Ω	æ	ø	
'30x	¿	¡	¬	√	ƒ	≈	Δ	«	"Cx
'31x	»	…	␣	À	Ã	Õ	Œ	œ	
'32x	–	—	"	"	'	'	÷	◊	"Dx
'33x	ÿ	Ÿ	⁄	¤	‹	›	fi	fl	
'34x	‡	·	‚	„	‰	Â	Ê	Á	"Ex
'35x	Ë	È	Í	Î	Ï	Ì	Ó	Ô	
'36x		Ò	Ú	Û	Ù	ı	ˆ	˜	"Fx
'37x	¯	˘	˙	˚	¸	˝	˛	ˇ	
	"8	"9	"A	"B	"C	"D	"E	"F	

◇ Yannis Haralambous
187, rue Nationale
59000 Lille
France
Fax: +33 20 91 05 64
yannis@gat.citilille.fr

Graphics

Addendum: A style option for rotated objects in LaTeX
***TUGboat* 13, no. 2, pp. 156–180**

Sebastian Rahtz and Leonor Barroca

In the cited article we stated "[T]he trigonometry macros ... are borrowed from psfig1.8; the original author is not credited there, so we cannot do so either."

The author has come forward: Phil Taylor. Credit where credit is due.

◇ Sebastian Rahtz
ArchaeoInformatica
12 Cygnet Street
York Y02 1AG
spqr@uk.ac.york.minster

◇ Leonor Barroca
Department of Computer Science
University of York
Heslington
York Y01 5DD
lmb@uk.ac.york.minster

Diag: A Drawing Preprocessor for LaTeX

Benjamin R. Seyfarth

Abstract

Diag is a preprocessor for drawing diagrams for LaTeX documents. The user prepares a text file containing commands in the `diag` language which are processed by `diag` producing `fig` commands which are then processed by `transfig` producing commands in a variety of formats acceptable to LaTeX. The `diag` preprocessor interprets a language with graphics commands using infix expressions with user-defined variables. In addition it provides a macro facility for simplifying repetitive operations. The combination of `diag` and `transfig` provides a simple, portable method for producing diagrams within LaTeX.

1 Introduction

The TeX typesetting system by Donald Knuth [3] provides a method for producing high-quality typesetting on a wide variety of computer systems. TeX has been augmented by Leslie Lamport's LaTeX [5] to provide an easier interface for TeX users. TeX was designed for typesetting text and mathematical formulas and does a splendid job for both. However, TeX provides nearly no support for graphics. LaTeX provides a variety of macro packages which do allow the user to incorporate drawings in a LaTeX document, but none of these is particularly easy to use. Diag provides a convenient alternative for LaTeX drawings.

LaTeX incorporates a simple picture drawing environment which can be used to produce lines, boxes, circles and arrows. Unfortunately, this package is designed around a set of line drawing characters which can only be used to draw lines with a limited number of preset slopes. In addition the package is written as TeX macros which means that specifications of x and y values can become onerous if the user must use TeX macros to compute locations. Lamport suggests that LaTeX drawings be completely designed using an initial drawing on a piece of graph paper. This is feasible, but it does not provide easily-modifiable diagrams.

There are several macro packages which have been written for LaTeX to provide better graphics. EPIC [7] is an extension of the LaTeX picture environment which uses the LaTeX drawing commands as primitives to produces lines, grids and arcs. EEPIC [4] is an extension of EPIC which uses `tpic` specials to overcome the limitations inherent in the LaTeX picture drawing primitives. The PiCTeX [8] package overcomes most of the limitations of the other macro packages and can produce high-quality graphics. Unfortunately it and the other macro packages suffer from the inconvenience of doing arithmetic using TeX macros.

In contrast to these LaTeX macro packages is the PIC preprocessor [2] for the `troff` typesetting system [6]. PIC provides a separate language supporting variables, infix expressions, looping and macros. This language allows a user to describe a diagram very simply using variables to define the x and y coordinates for graphics objects. This makes it easy to position objects relative to other objects which makes diagrams easier to modify.

There is a version of the PIC preprocessor called `tpic`, which has been altered to output TeX `\special` commands. These specials are then interpreted by DVI drivers to do the actual drawing. Unfortunately `tpic` is a modification of PIC and can only be distributed to licensed PIC users.

A completely different alternative for producing drawings in LaTeX is to use an interactive drawing program such as `fig` or `xfig`. `fig` is a graphics editor originally written by Supoj Sutanthavibul at the University of Texas. `xfig` is a version of `fig` written for the X Windowing System by Brian Smith of the Lawrence Berkeley Laboratory and others. Both these programs output `fig` commands which can be translated using `transfig` into EPIC, EEPIC, PiCTeX, `tpic` and several other formats usable in LaTeX. This is a convenient proposition for people with graphics terminals, but graphics terminals are not always available. Another drawback to using interactive drawing programs is that they do not generally support a convenient language interface. A language interface would allow users to write special programs to output graphic commands when there are many drawings to create.

The `diag` preprocessor provides a language similar to the PIC language, although it is considerably simplified. It does support variables, infix expressions, relative positioning and macros similarly to PIC. It does not presently support loops or `if` statements, nor does it support as many relative positioning options as PIC. It was decided that loops and conditional statements would be most useful for graphing mathematical functions and the author suggests using the `GNUPLOT` program for plotting functions. The relative positioning options in `diag` are fewer than those in PIC, but sufficient for most uses. The `diag` language is designed to be easy to learn and is capable of producing high-quality graphics for LaTeX documents.

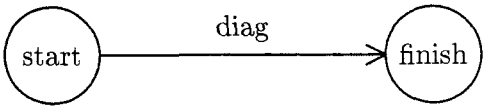


Figure 1: Getting There

2 Using diag

The input to diag is a text file containing diag commands. Let's suppose that we have a text file named "ex1.d" which contains the following:

```
Start: circle "start" at (1,1);
Finish: circle "finish" at (3,1);
arrow "diag" Start.e to Finish.w;
```

This file contains three diag commands. The first two draw circles and the third command draws an arrow from the start circle to the finish circle, placing the word "diag" above the arrow. The words before the colons of the first two commands are the names of the objects. The arrow is drawn from the easternmost point of the start circle to the westernmost point of the finish circle. We could modify the location of either circle and still connect the two circles using the same arrow command.

To convert the diag commands into PICTEX commands, we use the following command

```
diag < ex1.d | fig2dev -L pictex > ex1.tex
```

This will produce a file, "ex1.tex" containing commands which can be input into LATEX using

```
\begin{figure}
    \begin{center}
        \input{ex1}
    \end{center}
    \caption{Getting There}
\end{figure}
```

The resulting diagram is shown in Figure 1.

3 The Diag Coordinate System

The default coordinate system for diag uses measurements in inches. The origin of the coordinate system, $(0, 0)$, is defined to be the lower left corner of the diagram. From there increasing x values refer to points to the right and increasing y values refer to points up the page from the origin.

The default scaling can be altered by assigning a new value to the scale variable which is initially 1. Making scale larger will shrink your diagram, while making it larger will expand your diagram. It is possible to change scale in the middle of a diagram, but this is likely to cause confusion.

As graphics commands are executed, diag maintains a current drawing position. This position can be used as a default for most commands and it can be explicitly altered. The variables x and y refer to the current position and are available to the user.

4 The Diag Language

Parsing in diag is performed by an interpreter generated using the yacc parser generator. The interpreter consists of a lexical analyzer feeding the LALR(1) parser from yacc. The two work together to translate commands in the diag language into equivalent fig commands.

4.1 Diag Lexical Conventions

The lexical analyzer expands macros, ignores comments and groups input characters into lexical items. The macro expansion facility will be defined later. The diag lexical items are *identifiers*, *numbers*, *strings* and the following special characters:

```
+ - / * ( ) > .  ; :  ? and =
```

An identifier is a letter followed by any number of letters or digits. Upper and lower case letters are permitted and denote different identifiers. Several unexpected identifiers are keywords in diag and will most likely cause syntax errors if they are used as variable names. These include:

```
e n s w ne nw sw se
```

A number in diag must start with a digit and may have any number of digits afterwards with at most one decimal point. Any fraction less than 1.0 must have a leading zero as in "0.5". A number can be preceded by a minus sign.

A string is defined as in the C programming language to be anything between a pair of quote symbols as in "string". It is not possible to place a quote symbol in a diag string.

The special characters are used to form arithmetic expressions and for a handful of special purposes detailed below.

Comments in diag are identified by either a # or % and extend from that character to the end of the line. This allows comments to either stand alone or to be placed on the end of a command.

An identifier in diag is either a keyword or a variable name. A variable becomes defined either by an assignment statement or by a graphics command preceded by an identifier naming a graphics object. In either case the variable name is the first element of the command. Most commands start with one of the command keywords defined below and every diag command is terminated with a semicolon.

4.2 Diag Statements

A diagram is defined to be one or more statements in the `diag` language. Using Backus-Naur Form (BNF), we have:

diagram → *diagram statement ;*
→ *statement ;*

There are several types of statements in `diag`:

statement → *assignment*
→ *drawbox*
→ *drawcircle*
→ *drawellipse*
→ *drawline*
→ *drawarrow*
→ *drawtext*
→ *drawarc*
→ *drawcurve*
→ *gotostatement*

4.2.1 Assignment Statement

An assignment statement is defined to be a variable name followed by an equals sign and an arithmetic expression. The variable will be created if it does not already exist. The BNF for assignment statements and expressions is

assignment → *IDENTIFIER = expr*
expr → *IDENTIFIER*
→ *NUMBER*
→ *expr + expr*
→ *expr * expr*
→ *expr - expr*
→ *expr / expr*
→ *- expr*
→ *(expr)*
→ *IDENTIFIER . xory*
→ *IDENTIFIER . pos . xory*
xory → *x* | *y*

Precedence for arithmetic expressions follows the normal pattern with multiplication and division having higher precedence than addition and subtraction. Parenthetical expressions are evaluated first.

Boxes, circles and ellipses may be named within `diag`. This is done by preceding the command to draw an object by a variable name and a colon. Afterwards the object's name can be used to specify a position. The x and y coordinates of an object can be used in an expression by adding either "`.x`" or "`.y`" after the variable name.

There are eight compass point positions defined for every named object. These can be used to specify positions in a diagram. This can be quite convenient compared to computing a position like the northeast point of an ellipse.

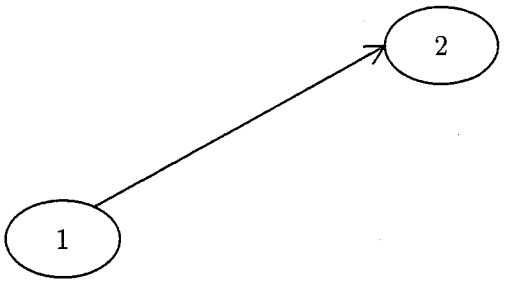


Figure 2: Ellipses and arrow

Object names and corner points are special cases of point expressions in `diag`. A point expression can also be specified as two arithmetic expressions in parentheses. Here is the syntax for point expressions:

ptexpr → *(expr , expr)*
→ *ptexpr + ptexpr*
→ *ptexpr - ptexpr*
→ *IDENTIFIER*
→ *IDENTIFIER . pos*
→ *IDENTIFIER . ?*
pos → *n* | *s* | *e* | *w* | *ne* | *nw* | *se* | *sw*

The following code draws two ellipses and connects them with an arrow:

```
x1 = 1;
x2 = x1 + 2;
e1: ellipse "1" at (x1,1);
e2: ellipse "2" at (x2,e1.y+1);
arrow from e1.ne to e2.w;
```

The diagram is in Figure 2. The second ellipse is placed two inches to the right and one inch higher than the first ellipse. The arrow is drawn from the northeast point of the first ellipse to the west compass point of the second ellipse.

Sometimes the eight compass points are not exactly the right points. Suppose we wish to draw an ellipse with four circles beneath and draw arrows to each circle. This is indicated with an object name followed by "`.?`" to indicate that `diag` should calculate a boundary point of the object for the connect line or arrow. This is shown in Figure 3. Here is the code required:

```
scale = 1.5;
e1: ellipse "Start" at (2.5,2);
c1: circle "1" at (1,1);
c2: circle "2" at (2,1);
c3: circle "3" at (3,1);
c4: circle "4" at (4,1);
arrow from e1.? to c1.?;
arrow from e1.? to c2.?;
arrow from e1.? to c3.?;
```

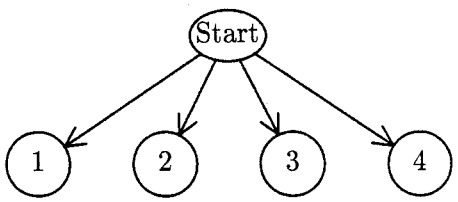


Figure 3: Ellipse and arrows to circles

```
arrow from e1.? to c4.?;
```

4.2.2 Drawing Boxes

A `diag` box is a rectangle which has sides parallel with the x and y axes. A box may include a text string placed at its center. A box may be described by giving two corner points or by specifying its height, width and center point. A box specified by corner points must specify two corners which must be opposite corners for the box. The syntax allows the keywords `from` and `to` to be optional.

A `box` command without two corner points specifies a box by height, width and center point. The predefined variables `boxht` and `boxwid` provide convenient defaults and the current point is used for the center if it is omitted.

drawbox	→	*objectname box boxopts*
boxopts	→	ϵ
	→	*boxopts from ptexpr to ptexpr*
	→	*boxopts label*
	→	*boxopts height expr*
	→	*boxopts width expr*
	→	*boxopts invisible*
	→	*boxopts at ptexpr*
objectname	→	ϵ | *IDENTIFIER :*
label	→	*STRING*
from	→	ϵ | *from*
to	→	ϵ | *to*
at	→	ϵ | *at*

Notice that a box can be `invisible`. This can be useful for drawing lines between words in a parse tree. Consider the following code and its diagram in Figure 4:

```
scale = 1.5;
boxht = 0.3;
b1: box invisible width 1.5 "sentence"
    at (2,2);
b2: box invisible width 1 "subject"
    at (1,1);
b3: box invisible width 1 "verb"
    at (2,1);
b4: box invisible width 1 "object"
    at (3,1);
line from b1.? to b2.?;
line from b1.? to b3.?;
line from b1.? to b4.?;
```

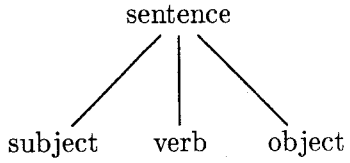


Figure 4: Simple sentence

4.2.3 Drawing Circles and Ellipses

A `diag` circle is defined by its radius and center point. The default for the radius is provided by the variable "`circlerad`", while the center point defaults to the current drawing position. An ellipse is defined similarly except that an ellipse has a major axis and a minor axis rather than a radius. In `diag` the major axis always refers to the x axis of an ellipse and the minor axis refers to the y axis. The defaults for the ellipse axes are the variables "`majoraxis`" and "`minoraxis`".

drawcircle	→	*objectname circle circleopts*
circleopts	→	ϵ
	→	*circleopts label*
	→	*circleopts radius expr*
	→	*circleopts at ptexpr*
drawellipse	→	*objectname ellipse ellipseopts*
ellipseopts	→	ϵ
	→	*ellipseopts label*
	→	*ellipseopts major expr*
	→	*ellipseopts minor expr*
	→	*ellipseopts at ptexpr*

4.2.4 Drawing Lines and Arrows

There are two basic ways to draw lines and arrows. First you can specify the start and end points for the line. The keyword `from` is optional, but the keyword `to` is required if the end point is specified. The second way to specify a line is to specify the end point, a direction and a line length. The only possible directions are up, down, left and right. The start point defaults to the current point and the direction defaults to right.

Text may be drawn at the midpoint of the line or arrow. Following a text string you may optionally specify where to place the text relative to the midpoint of the line. The default is to place the text slightly above the midpoint.

Here is the full syntax for line and arrow drawing:

drawline → *line lineopts*
lineopts → ϵ
→ *lineopts label where*
→ *lineopts from ptexpr*
→ *lineopts to ptexpr*
→ *lineopts direction expr*
direction → *up* | *down* | *left* | *right*
drawarrow → *arrow lineopts*
where → ϵ | *above* | *below*
→ *left* | *right*

4.2.5 Placing Text at Arbitrary Positions

It is possible to place a text string at any arbitrary position of a diagram by entering the string followed by the position for the string. The keyword `at` is optional and the position defaults to the current position.

drawtext → *STRING at ptexpr*
→ *STRING*

4.2.6 Drawing Circular Arcs and Curves

There are two types of curves supported by `diag`. They are both circular arcs, but they are given separate commands for simplicity. The first type of circular arc is drawn with the `arc` command. It is always a 90 degree arc in one of the four quadrants. Such an arc is specified by starting point, quadrant and direction. The starting point defaults to the current point. The quadrant is specified as `ur`, `ul`, `ll` or `lr` to indicate upper-right, upper-left, lower-left or lower-right. The direction is either `cw` or `ccw` to indicate clockwise or counter-clockwise.

The second type of arc is specified by start point, end point and curvature. The curvature is specified by the keyword `bend` followed by a number or an expression. The curvature defaults to value of the variable `curvature`. In general, the curvature should be between 0 and 1, but not very close to either.

The `bend` keyword is illustrated in Figure 5. In this figure the `bend` was set to 0.2. This means that the height of the curve, h, is 0.2 times the length of the chord c. The code to produce Figure 5 is:

```
scale = 2.0;
curve cw bend 0.2 (1,1) to (5,1);
linestyle = dashed;
line "c" below from (1,1) to (5,1);
line "h" left from (3,1) to (3,1.8);
```

It is also possible to place an arrow head on the end point of a curve. This is done by placing a greater than symbol in the command. The full syntax for drawing arcs and curves is:

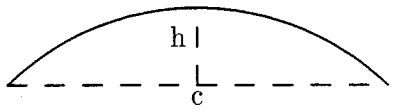


Figure 5: Curvature definition

drawarc → *arc arcopts*
arcopts → ϵ
→ *arcopts quadrant*
→ *arcopts arcdir*
→ *arcopts radius expr*
→ *arcopts ptexpr*
drawcurve → *curve curveopts*
curveopts → ϵ
→ *curveopts label where*
→ *curveopts* >
→ *curveopts arcdir*
→ *curveopts bend expr*
→ *curveopts from ptexpr*
→ *curveopts to ptexpr*
arcdir → *cw* | *ccw*
quadrant → *ul* | *ur* | *ll* | *lr*

4.2.7 Changing the Current Drawing Position

There is a `goto` command to explicitly change the current drawing position of `diag`. It consists of the keyword `goto` followed by an arbitrary point expression.

gotostatement → *goto ptexpr*

4.3 Predefined Variables

There are a number of variables created by `diag` which can be changed to control things like line thickness and arrow head length. These variables are different from user-created variables only in the sense that they exist when `diag` starts and `diag` uses their values for various purposes.

4.3.1 arcrad and circlerad

The `arc` command will draw a 90 degree arc of a certain radius. If the radius is not specified, it will default to the value of `arcrad`. Similarly the circle command defaults to a radius of `circlerad`. Both these variables are measurements in inches unless `scale` has been changed. The initial values for `arcrad` and `circlerad` are each 0.25 inches.

4.3.2 boxht and boxwid

The `box` command can be used to draw a box with a center at a certain position. In that usage the user can specify the height and width of the box, or allow `diag` to use `boxht` and `boxwid` as default values.

These are measurements in inches by default. The initial value for `boxht` is 0.5 inches and the initial value for `boxwid` is 0.75 inches.

4.3.3 curvature

The `curve` command allows the user to specify the curvature using the `bend` keyword. If `bend` is not specified, the value of `curvature` will be used instead. The initial value for `curvature` is 0.2.

4.3.4 dashlength

If the linestyle has been selected as dashed or dotted, then the variable `dashlength` can be set to control the length of dashes or the spacing of dots. This is a measurement in inches by default. The initial value of `dashlength` is 0.1 inches.

4.3.5 head

The length of the head of an arrow can be controlled by the `head` variable. This is a measurement in inches by default. The initial value of `head` is 0.1 inches.

4.3.6 linestyle

The variable `linestyle` can be used to change the line style from `solid` to `dashed` or `dotted`. The initial value of `linestyle` is `solid`. The variables `solid`, `dashed`, and `dotted` have the values 0, 1 and 2 matching their `fig` values.

4.3.7 linethickness

This variable controls the thickness of lines in pixels. Its initial value is 5 pixels.

4.3.8 majoraxis and minoraxis

These variables are used as defaults by the `ellipse` command. The x axis is always considered the major axis and the the y axis the minor axis. These variables represent inches by default. The initial value for `majoraxis` is 0.3 inches and the initial value for `minoraxis` is 0.2 inches.

4.3.9 pi

This variable has the value 3.14159 and should not be changed.

4.3.10 scale

The default scaling of coordinates in `diag` is in inches. This can be overridden by assigning a new value to `scale`. Coordinates in `diag` are divided by `scale` before translating into dot positions on the page. This means that making `scale` greater than 1.0 will shrink the diagram.

5 Defining and Using Macros

The macro facility of `diag` is implemented as a text replacement algorithm by the lexical analyzer. A macro is defined by the keyword `define`, the name of the macro, and its replacement text. The replacement text is identified by starting and ending it with a special symbol such as `%`.

A macro invocation is either the macro name followed by a semicolon or the name followed by parameters in parentheses. These parameters are positional parameters separated by commas and are referred to within the macro's replacement text as `$1`, `$2`, ..., `$9`. There can be up to nine positional parameters.

A sample macro definition to define a macro to draw three boxes centered at a given position would be

```
define ThreeBoxes #
   box at ($1-boxwid,$2);
   box at ($1,$2);
   box at ($1+boxwid,$2);
#
```

This macro be used to create a three by three arrangement of boxes using

```
p = 2;
ThreeBoxes ( 3, p );
ThreeBoxes ( 3, p-boxheight );
ThreeBoxes ( 3, p+boxheight );
```

6 A Larger Diagram

In this section a diagram is shown of an array of pointers to structures containing pointers and names. This is a reasonable example for illustrating macros. The code for this example is split up into several sections along with some explanation. The diagram is shown in Figure 6.

First there is a macro to draw a `NULL` pointer to the right of some of the array elements. This macro uses the variable `bw` to determine how long the constituent lines should be.

```
define Null %
   line right bw * 1.0;
   line down bw * 0.15;
   X = x;
   Y = y;
   line (X-0.2*bw,Y) to (X+0.2*bw,Y);
   line (X-0.125*bw,Y-0.05*bw)
     to (X+0.125*bw,Y-0.05*bw);
   line (X-0.05*bw,Y-0.1*bw)
     to (X+0.05*bw,Y-0.1*bw);
%
```

Next there is a macro to draw a box and then move down `bw` inches. This macro draws the box

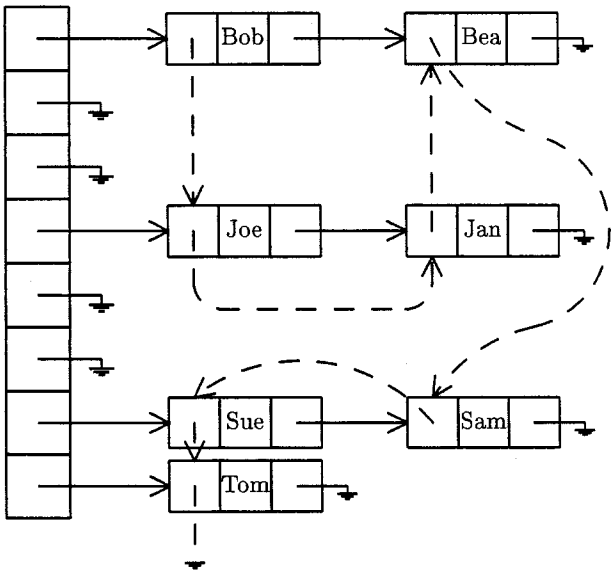


Figure 6: Symbol Table with an Auxiliary Linked List

at (x1,y1) and then modifies y1 to prepare for the next vertical box.

```
define Vbox %
   box (x1,y1) to (x1+bw,y1-bw);
   y1 = y1 - bw;
   goto (x1+bw/2,y1+bw/2);
%
```

Similarly there is a macro to draw a horizontal box at (x1,y1) and move to the right bw inches. This macro is used to draw structures containing three boxes. In the struct macro each Hbox is given a name to make it easy to connect the components later. The first parameter of struct is the prefix for the names of the boxes. The middle box is given that name and the others are given that name with an added l or r.

```
define Hbox %
   box $1 (x1,y1) to (x1+bw,y1-bw);
   x1 = x1 + bw;
   goto (x1+bw/2,y1+bw/2);
%

define struct %
   x1 = $2 - bw * 1.5;
   y1 = $3 + bw * 0.5;
   $1l: Hbox;
   $1: Hbox ( $4 );
   $1r: Hbox;
%
```

Now begins the first non-macro code. First the variables x1, y1 and bw are initialized and then the array of pointers to records is drawn along with several null pointers.

```
scale = 1.8;
x1 = 1;
y1 = 4;
bw = 0.6;
#
# Place the array of record pointers
# on the left.
#
A: Vbox;
B: Vbox;
Null;
C: Vbox;
Null;
D: Vbox;
E: Vbox;
Null;
F: Vbox;
Null;
G: Vbox;
H: Vbox;
```

Next the variable bw is shrunk to make slightly smaller boxes and then the structures are drawn. After the structures on a row are drawn, connecting arrows and null pointers are drawn.

```
bw = bw * 0.8;
#
# Add the 'B' records
#
struct(s1,3.25,A.y,"{\small Bob}");
struct(s2,5.5,A.y,"{\small Bea}");
arrow from A to s1l.w;
arrow from s1r to s2l.w;
goto s2r;
Null;
#
# Add the 'J' records
#
struct(s3,3.25,D.y,"{\small Joe}");
struct(s4,5.5,D.y,"{\small Jan}");
arrow from D to s3l.w;
arrow from s3r to s4l.w;
goto s4r;
Null;
#
# Add the 'S' records
#
struct(s5,3.25,G.y,"{\small Sue}");
struct(s6,5.5,G.y,"{\small Sam}");
arrow from G to s5l.w;
arrow from s5r to s6l.w;
goto s6r;
Null;
```

```
#
# Add the 'T' record
#
struct(s7,3.25,H.y,"{\small Tom}");
arrow from H to s71.w;
goto s7r;
Null;
```

Finally we add a collection of dashed arrows and curves to indicate another linked list comprising the same set of records.

```
#
# Add auxiliary linked list pointers
#
linestyle = dashed;
arrow from s11 to s31.n;
goto s31;
line down bw;
arc ll ccw;
line right s4.x-s3.x-2*arcrad;
arc lr ccw;
arrow to s41.s;
arrow s41 to s21.s;
ya = (s2.y+s4.y) / 2;
yb = (s4.y+s6.y) / 2;
curve ccw bend 0.1 from s21
   to (s4r.x,ya);
curve cw bend 0.4 to (s4r.x,yb);
curve > ccw bend 0.1 to s61.n;
curve > ccw bend 0.2 from s61 to s51.n;
arrow from s51 to s71.n;
goto s71;
line down bw*1.5;
X = x;
Y = y;
linestyle = solid;
line (X-0.2*bw,Y) to (X+0.2*bw,Y);
line (X-0.125*bw,Y-0.05*bw)
  to (X+0.125*bw,Y-0.05*bw);
line (X-0.05*bw,Y-0.1*bw)
  to (X+0.05*bw,Y-0.1*bw);
```

7 Possible Additions to the Language

The most obvious features missing from the language are conditional statements and loops. These would clearly be useful, but are not critical for the anticipated uses of `diag`.

If `diag` needs conditional statements and loops at some future date, it is likely that procedures would also be added. The current implementation uses macros which look like procedure calls, but they use global variables. There is no such thing as a local variable and using macros which manipulate variables is a hazard. Procedures would eliminate this problem.

Another possible improvement to `diag` would be to use the ".?" positioning operator with the `curve` command. This would be relatively easy to implement and could be useful for some diagrams.

It would be useful to draw and fill polygons. This is supported by `transfig` and would be easy to add to `diag`. Another feature supported by `transfig` which could be added is a spline drawing command.

There are many more features which could be added to the `diag` language. The author selected the most basic commands for the first version of `diag`. It is anticipated that the language will grow as needs arise.

References

[1] Micah Beck, *TransFig: Portable Figures for TEX*, Cornell University Dept. of Computer Science Technical Report #89-967, February 1989.

[2] Brian W. Kernighan, *PIC - A Graphics Language for Typesetting*, Bell Laboratories Computing Science Technical Report 85, March 1982.

[3] Donald E. Knuth, *The TEXbook*, Addison-Wesley, 1986.

[4] Conrad Kwok, *EEPIC: Extensions to* EPIC *and LATEX Picture Environment*, Software documentation, University of California, Davis, Dept. of Computer Science, July 1988.

[5] Leslie Lamport, *LATEX: A Document Preparation System*, Reading, Mass.: Addison-Wesley, 1986.

[6] Joseph F. Osanna, *NROFF/TROFF User's Manual*, Bell Laboratories Computing Science Technical Report 54, October 11, 1976.

[7] Sunil Podar, *Enhancements to the Picture Environment of LATEX*, State University of New York at Stony Brook, Dept. of Computer Science, Technical Report #86-17, July 1986.

[8] Michael Wichura, *The PICTEX Manual*, University of Chicago, November 1986.

◇ Benjamin R. Seyfarth
Department of Computer Science and Statistics
The University of Southern Mississippi
Southern Station
Box 5106
Hattiesburg, Mississippi 39406-5106

Book Reviews

Review of: **TEX for the Beginner**

Victor Eijkhout

Wynter Snow, *TEX for the Beginner*. Addison-Wesley, 1992. xii + 377 pp. + index (23 pp.) ISBN 0-201-54799-6.

The spectrum of books about TEX has been broadened by another beginner's book about plain TEX: with *Introduction to TEX* (Norbert Schwarz; Addison-Wesley, 1990), *A Beginner's Book of TEX* (Raymond Seroul and Silvio Levy; Addison-Wesley, 1991) and *TEX by Example* (Arvind Borde; Academic Press, 1992) there is now *TEX for the Beginner* by Wynter Snow.

In contrast to the Seroul and Levy book, which is somewhat academic in style, this book has a very chatty style that is both its strong and its weak point. The author has a lively style of writing and uses rather unorthodox metaphors for TEX's behaviour (did you know that macros come in meat-and-potatoes and jelly-doughnut varieties?), which in general are quite illuminating.

For instance, the fact that lines are spaced at `\baselineskip` distance is described "as if TEX has a ruler exactly `\baselineskip` long that it uses to decide where the next shelf should go", which is a good way of putting it. Sometimes, however, too much American cultural background is needed. For instance, I think I understand what is meant by "back in grade school" (page 90), but I'm not sure what is meant by "a taffy-like substance" (page 97). Considering how international the TEX community is, this point is not without significance.

1 Structure of the book

TEX for the Beginner is divided in five parts. The first two parts are about the basics of using TEX. In particular the first part is very practically oriented: there are tips for people using word processors who dump their file in ascii mode, and there are instructions for people using the Macintosh implementation of TEX, *Textures*.

Part III is the longest and probably the most useful. Its sixteen chapters explain TEX commands and give macros for the most common things that you do in TEX. Part IV is about advanced topics such as boxes and rules, and it has a short chapter about more theoretical aspects of TEX. Part V is about 'Bug Diagnosis' and has two appendices.

The book's index contains both the concepts and control sequences that were treated in the book itself, as well as the TEX primitives and plain TEX control sequences that have to be looked up in *The TEXbook*. The latter kind simply do not have page references. This makes sense: if you are trying to decode someone else's macros it tells you that something is a primitive or part of plain TEX (and you'll have to consult another book), and it can prevent you from redefining such commands.

Although the division in parts globally makes sense, on a detailed level the organization is somewhat messy. For instance, the chapter 'Adjusting awkward line breaks' is in part I among the basics, while 'Adjusting awkward page breaks' is in part IV with advanced topics. Also, the 'actor model', the author's metaphor for TEX's workings, is introduced in a chapter called 'Getting a printout'. While its place is logical in the course of the exposition, it makes for awkward referencing.

2 Who is this book meant for?

Among the books that aim to be for the beginning TEXer, this book presents the most basic information, including tidbits that are specific to certain computers. The discussion is clearly for people wanting to learn plain TEX, but there are so-called 'LATEX notes', which can function both to point the plain TEX user to corresponding concepts in LATEX, and to help the LATEX user make a transition to plain TEX. They are not enough to learn LATEX from. Too often they tell the reader that the whole of a chapter cannot be used in LATEX.

Another concept that will help the beginner is that the author identifies a few dozen bugs, which are clearly marked in the text, and referenced in the index. They have been given names with varying degrees of helpfulness, ranging from the 'Disappearing footnote bug' to the 'Cart before the horse bug'.

3 Level of complexity

Since this is a beginner's book, some topics are only touched on lightly. The author advises the reader to write lots of macros (and gives many examples) but real TEX programming is never done: the control sequences `\edef` and `\expandafter` don't appear in the book.

Similarly, there are chapters about headers and footers and about (stationary and floating) figures, but `\output` is not mentioned.

4 Layout

Unfortunately, I cannot say much positive about the looks of this book. The author uses a liberal amount of white space around examples and list items, the text has a ragged right margin, and the output of the examples is indented to a different margin than the input code. All together it swims before my eyes. In addition to changing from roman type to typewriter for the examples, the author uses Computer Modern Sans Serif type for the bugs and the LaTeX notes. This only adds to the confusion. The fact that headings are underlined with a page-wide rule makes the pages look only slightly more structured.

The back cover of this book appears to use the Computer Modern type, and it has plain TeX's extended spacing after punctuation, but strangely enough the TeX and LaTeX logos do not have any of their characteristic back kerning, and ligatures are missing. Let's say the front cover makes up for the back.

5 But are there jokes?

Snow has a lively imagination in coming up with novel analogies for TeX's workings, and that definitely makes this book amusing to read. And if the author has ever been teased about her name, she certainly hasn't suffered from it: the parts of the book have ski-oriented titles ('Onto the slopes', 'Down we go'), and the front cover shows lion cubs throwing snowballs and building a snow man in the likeness of the TeX lion.

6 Evaluation

TeX for the Beginner goes easy on the theory of TeX, and instead takes the reader by the hand, showing in a practical way how to get the work done. The explanations are enough to give a beginning TeX user a basic understanding, but are often not 100% accurate. For instance, the statement 'The primitives `\csname` and `\endcsname` take a string of characters and convert it into a control sequence' and its clarification in a footnote '`\csname` converts a token list into a control sequence' are both slightly off: in between `\csname` and `\endcsname` all tokens are expanded (so unexpandable tokens are not allowed) until only character tokens remain, then a control sequence is made out of these.

However, I haven't found any real errors in the book.

If the explanations can best be characterized as 'adequate', similarly the macros in part II get the user under way, but they are not complete. For instance, chapter 16, 'Obeying lines and spaces: verse and computer code' does not give macros for a true verbatim mode, so the computer code that can be handled is rather limited. (The examples are in the language Logo, and don't use special characters such as the caret, which appears in Pascal, or braces, which occur very often in C.)

More surprising, chapter 17 about headings never mentions `\nobreak` to prevent a page break between a heading and subsequent text. A `\TitleSection` example in the chapter about headers and footers does incorporate a `\nobreak`, however, without drawing attention to it.

In general, though, the examples cover a lot of territory, and they do so with a good explanation.

Snow introduces some non-standard terminology ('giving a reporter an input' instead of 'giving a parameter a value') which may make the transition to more advanced TeX literature harder for the reader. Since the terminology is used consistently I don't have too much of a problem with it.

In all, this book is a useful addition to the TeX library. It is well written and contains many practical examples. Readers who would like to learn TeX in a linear, incremental way, should definitely check out this book.

◇ Victor Eijkhout
Department of Computer Science
University of Tennessee at Knoxville
Knoxville TN 37996-1301
Internet: `eijkhout@cs.utk.edu`

Review of: TeX by Example: A Beginner's Guide

George D. Greenwade

Arvind Borde, *TeX by Example: A Beginner's Guide*. San Diego: Academic Press, 1992. xiv + 169 pp. ISBN 0-12-117650-9. $19.95.

Arvind Borde's *TeX by Example: A Beginner's Guide* is very obviously a labor of love in introducing new users to the world of TeX — or the result of the frustration associated with having to repeatedly explain commonly-used formatting commands to one's colleagues. Since it is noted that the book

represents a significantly expanded version of the author's widely-distributed *An Absolute Beginner's Guide to Using TEX* (1987), either of these motivations is possible as a non-unique factor. The text utilizes a practical and visually appealing side-by-side layout, making Borde's book one of the most useful introductory primers on the market for virtually any language. Notably, most of this is achieved without the employment of extensive macros; however, where non-standard TEX or author-written macros are employed, it is liberally noted.

Does this place *TEX by Example* in the "introductory" class of TEX materials? Very decidedly so! However, readers should not make the mistake of placing this innovative text in the "introductory-only" class of TEX materials. Indeed, in the case of *TEX by Example* it is not wise to limit the readership even to a "TEX-only" audience as users of any derivative of TEX — LATEX, AMS-TEX, REVTEX, eplain, ... — can benefit from the information and examples contained in it by reviewing the underlying command structure used in a "plain" TEX environment to generate various outputs. Since derivatives may or may not be fully compatible with all aspects of "plain" TEX, the examples may not be directly applicable, although the concepts certainly provide an acceptable template for extensions.

The real beauty of *TEX by Example* lies in its design. From the beginning of the text (p. vi) to the end (p. 91), each right-hand side (RHS) page is TEX output, covering properly-formatted topics from poetry, to philosophy, to physics, to mathematics, to the game of cricket, to just about anything else — even TEX.

Each facing left-hand side (LHS) page includes two very important components. First, the actual TEX input code which created the TEX output on the RHS facing page is provided verbatim. Therefore, a very quick reference tool is at the fingertips of the reader — if you see what you want the final output to look like on a RHS page, just look directly to the facing LHS page and the necessary TEX code to generate this output is immediately available. Second, the LHS input pages are marvelously documented in a series of footnotes which generally explain some of the differences available to users, as well as how the structure of the input can impact the ultimate output. If a topic has been covered previously, the reader is usually provided at least a hint where to look earlier in the text for its previous discussion.

Additionally, *TEX by Example* includes at least two other components of interest. First, following the text of the book (pp. 91–159) is a relatively comprehensive alphabetical Appendix, which serves as both an index of topics and a glossary of commands. It is fashioned very much after *The Permuted Index of TEX Commands*. Admittedly, this is not the best way to learn a language; however, many basic aspects of TEX are explained throughout this Appendix. Also, the Appendix serves as a good quick reference which can be used in concert with other manuals. The reader can easily look for a topic, such as "equation numbers" and be pointed to examples from within the text, as well as related commands. Alternately, the reader can just as easily look for a TEX command, such as "`\medbreak`", and find an explanation of that command's use. A very important dimension of the glossary of commands is the clear identification of TEX primitives, as compared to TEX composite commands or command macros utilized directly in the production of the book.

Second, Borde has been generous enough to include virtually all of his macros used in *TEX by Example* for use by the reader and public at large. The major command macros are referenced on page 136 of the Appendix, which points the user to their use and first appearance in the text. One of the most intriguing commands defined by Borde is `\fermat`. This command tackles Pierre Fermat's generally accepted "theorem", making necessary calculations, then reporting the result. The insights from this reveal TEX's usually-undiscussed capability as a programming language.

The book ends with its Epilogue. The Epilogue includes the major layout commands used by Borde in his creation. The complete macros used in the text appear here. Readers can easily copy them for their own use, selectively use them as templates for their own macro sets, or merely refer to them for some of the tricks Borde employs (such as the side-by-side layout with accompanying notes). If the reader doesn't wish to type in the text of these macros, they have been made available from most major TEX-related archives for electronic mail retrieval or access through anonymous ftp, or, in the absence of an electronic link, the files are available on diskette.

In summary, users at all levels of TEX-related processing languages ought to have this innovative guide handy. While covering a wide array of topics, the general tone of the text is very reader friendly, which allows it to serve as a remarkable beginner's guide, per its title. However, its comprehensive overview of usage, meaning, and structure is invaluable to users at all levels, making it a

very powerful reference guide to complement the standard manuals for your favorite flavor of TEX.

References

Arvind Borde, *An Absolute Beginner's Guide to Using TEX*, informal report (prepared for the Syracuse University Relativity Group), 1987.

Bill Cheswick, *A Permuted Index for TEX and LATEX*, Providence, TEX Users Group, *TEXniques*, No. 14, 1991.

◇ George D. Greenwade
Department of Economics and Business Analysis
College of Business Administration
P. O. Box 2118
Sam Houston State University
Huntsville, TX 77341-2118
bed_gdg@SHSU.edu

Review of: **Desktop Publishing in Astronomy & Space Sciences**

A. G. W. Cameron

André Heck, ed., *Desktop Publishing in Astronomy & Space Sciences*. Singapore: World Publishing Co. Pte. Ltd., 1992. ISBN 981-02-0915-0.

These are the proceedings of a conference held on 1–3 October 1991 at the Astronomical Observatory, Strasbourg, France. This meeting was organized by André Heck, who edited the proceedings. The book was produced in record time: I am writing this review in September 1992.*

The authors of the individual articles prepared them in camera-ready form, most of them using TEX or LATEX. However, I am afraid that the resulting product is not a publishing work of beauty. The instructions to the authors were to prepare a manuscript at 12 points with a baselineskip of 14 points, and I presume that the hsize was 6.5 inches and the vsize was 9 inches. But as published the hsize was 4.6 inches and the vsize was 6.5 inches. With this reduction I had to use a magnifying glass to read some of the abstracts, which were prepared at less than 12 points, and if the original text was less than 12 points I usually had to use a magnifying glass for that too. There is a name index but no subject index. Despite these problems this book is well worth the attention of TEX devotees.

The articles on TEX and LATEX will contain no real surprises for readers of TUGboat, as they could have appeared in the proceedings of the TUG annual meeting for 1991 which was devoted to publishing. The leading journals in astronomy and astrophysics are now encouraging manuscript submission in TEX or LATEX. For me the fun part of the book was in reading what those who do not use TEX had to say. Some of the time it was clearly due to ignorance. For example, J. O. Breen gave a talk which was an undiluted advertisement for the Apple Macintosh and the wonderful software that can run on it, including this gem: "equation editors such as Prescience's Expressionist and Design Science's MathType, create typeset equations to be pasted into scientific and technical papers." Mr. Breen is not an editor, nor is he a publishing astrophysicist as far as I can discover.

Mr. Breen prefers to create manuscripts using WordPerfect 5.1, as do a couple of other contributors, and output examples of this word processor are on display. TEX users can justifiably feel smug and optionally can wear a condescending sneer. WP 5.1 has terrible internal spacing in its equations and displays extremely loose lines, showing far too much white space. The other non-TEX contributors do not identify their output programs, but their contributions give good illustrations of why desktop publishing acquired a bad name in its early stages.

There was a significant amount of discussion about how the software should evolve. There was clearly a desire on the part of some people that a wider selection of fonts should be readily available with TEX. For example, publishers frequently would like to use Times Roman; at least one of them converts to Times after receiving a compuscript in Computer Modern, but others are clearly prepared to use Computer Modern. I was surprised to find essentially no discussion of scalable fonts; for some time now I have been using such fonts in conjunction with Vector TEX from MicroPress, and I don't know how I ever got along without them previously. People would like to have better graphics facilities to use with TEX, either as a language facility incorporated into TEX, or a more convenient way to merge PostScript into a TEX file.

There is a general expectation in the astronomical community that on-line electronic publishing will have a rapid growth in the near future. Editors clearly intend to subject papers for electronic

---

* Editor's note: The review copy was received at the TUG office in June 1992.

publication to a standard refereeing process. But what format should such papers have? Since people will want to be able to print copies of particular papers, formatting in TEX becomes a leading possibility. Such papers could then be passed through a standard TEXing process and printed, or they could be read after processing through a TEX screen previewer. However, there was additional discussion about how one could create electronic archives consisting of large data bases of electronic papers. Would TEX help in the archiving and accessing processes?

It has been very interesting to read a book about desktop publishing in which TEX is the leading contender, rather than being out of sight among the packages that allow you to produce fancy newsletters. But the growth of desktop publishing software packages has been so rapid that it is clear that TEX must make good use of the new emerging technologies if it is to maintain its leading position among astronomers and other scientific users. Despite its faults of presentation, the book is recommended.

◇ A. G. W. Cameron
Harvard-Smithsonian Center for Astrophysics
60 Garden Street
Cambridge, MA 02138

Typesetting on PCs

TEX-386 implementations for IBM PCs: Comparative timings

Erich Neuwirth

Timing tests were performed on several implementations of TEX for IBM PCs (and compatibles). All currently available 386-specific implementations were tested. Additionally the latest available versions of emTEX for other processor classes were tested. Since the last published benchmarks used sbTEX in the 8086 version as the standard the latest version of this implementation was also included for reference purposes. PCTEX as the most widely used commercial implementation of TEX for PCs was used in its 8086 flavor for the same reason.

These were the tested versions:

SBTEX 3.8
emTEX 3.0 [3a]
emTEX286 3.0 [3a]
bigemTEX 3.0 [3a]
bigemTEX286 3.0 [3a]
PCTEX 3.14
emTEX386 (beta8)
TEXas 1.0
PCTEX386 3.14
BigPCTEX386 3.14

The files used for testing were the same as in the previously published series of benchmarks:

- Text1 is *The TEXbook.** It is 494 pages long.
- Text2 is a mathematical paper which needs LATEX and PICTEX, so it really uses lots of memory. The document is 11 pages long.
- Text3 is a book of solutions for a college mathematics textbook. It consists almost completely of formulas and there is almost no text. It is among the most complicated TEX files I have ever seen. It uses LATEX and additionally the `msxm` and `msym` fonts from (old) $\mathcal{A}\mathcal{M}\mathcal{S}$-TEX. The document is 40 pages long.
- Text4 is the demo file for MusicTEX, which is a rather large macro package for typesetting music. The document is 2 pages long.
- Text5 is Michael Wichura's original paper from *TUGboat* 9, no. 2, describing PICTEX. It makes extensive use of PICTEX macros and also uses rather large data sets for the graphics. Additionally it uses the *TUGboat* macro files (in a stripped down version). The document is 10 (narrower than a page) columns long.
- Text6 is Barbara Beeton's review and the Boston Computer Society mathematical text processor benchmark from *TUGboat* 6, no. 3. It (naturally) contains complicated formulas and uses the *TUGboat* style. The document is 4 pages long.

Table 1 shows the times associated with the tests.

The following special events occurred during the benchmark:

(1) capacity exceeded, program stopped.
(2) Michael Wichura's PICTEX article could only be run in one column mode with non-386 versions of TEX having standard TEX memory.

* The file for *The TEXbook* used with permission of the American Mathematical Society.

If we take emTEX 386 as the base for a comparison of performance, we get relative indices of performance as shown in Table 2 (a low value indicates fast performance).

All these benchmark runs were performed on a 50MHz 486 DX2 machine with MSDOS 5.0 installed. All programs were run with QEMM-386 installed giving 600KB free main memory and either 7Mb of extended memory or 7Mb of simulated expanded memory.

When using 386 machines most people do not run just plain MSDOS. Additionally in most cases multitasking environments are used. Since all 386 specific TEX implementations are using some sort of DOS extender it is important to know which version of TEX will be compatible with which multitasking environment. Experience has taught us that one must be very careful when using multitasking environments. Therefore some additional tests were performed. All tested implementations of TEX were run under DesqView 386 and under Windows 3.1 in standard and in enhanced mode. To reduce the number of runs only text3 (being a very complicated TEX document) was used for this test.

Table 3 gives an overview of timings and of compatibility problems.

Table 1. Test times

	Text1	Text2	Text3	Text4	Text5	Text6
SBTEX	2:39	(1)	0:33	0:10	0:18 (2)	0:08
emTEX	2:50	(1)	0:35	0:10	0:18 (2)	0:09
emTEX286	2:43	(1)	0:34	0:10	0:18 (2)	0:09
bigemTEX	6:20	0:27	1:35	0:24	0:40	0:21
bigemTEX286	6:00	0:26	1:31	0:23	0:39	0:21
PCTEX	3:11	0:15	0:42	0:12	0:23	0:11
emTEX386	1:41	0:09	0:22	0:08	0:13	0:07
TEXas	1:49	0:15	0:30	0:12	0:20	0:11
PCTEX386	1:44	0:10	0:22	0:08	0:15	0:07
BigPCTEX386	1:48	0:10	0:23	0:08	0:16	0:07

Table 2. Relative performance

	Text1	Text2	Text3	Text4	Text5	Text6
SBTEX	1.57		1.50	1.25	1.38	1.14
emTEX	1.68		1.59	1.25	1.38	1.29
emTEX286	1.61		1.55	1.25	1.38	1.29
bigemTEX	3.76	3.00	4.32	3.00	3.08	3.00
bigemTEX286	3.56	2.89	4.14	2.87	3.00	3.00
PCTEX	1.89	1.67	1.91	1.50	1.77	1.57
emTEX386	1.00	1.00	1.00	1.00	1.00	1.00
TEXas	1.08	1.67	1.36	1.50	1.54	1.57
PCTEX386	1.03	1.11	1.00	1.00	1.15	1.00
BigPCTEX386	1.07	1.11	1.05	1.00	1.23	1.00

Table 3. Test times and compatibility with Windows and DesqView

	DesqView 386	Windows standard	Windows enhanced
SBTEX	(3)	(3)	0:33
emTEX	0:36	0:34	0:35
emTEX286	0:34	0:34	0:33
bigemTEX	6:37	5:40	2:29
bigemTEX286	6:24	5:30	2:20
PCTEX	0:56	1:22	0:54
emTEX386	0:23	3:38	(5)
TEXas	0:33	(4)	0:32
PCTEX386	0:25	(6)	0:24
BigPCTEX386	0:26	(6)	0:25

The following problems occurred during the tests:

(3) Program started, but not enough memory to compile document.
(4) Loader failed (DOS4GW)
(5) DPMI not supported
(6) Insufficient physical memory available

This table clearly shows that none of the available implementations of TEX for 386 PCs under DOS can run in both variants of Windows: emTEX cannot run in enhanced mode, and PCTEX386 and TEXas cannot run in standard mode. Since Windows only functions as a full multitasker for DOS tasks in enhanced mode, running a 386-TEX for this configuration can be accomplished only with PCTEX and TEXas. When using DesqView all three 386-TEX implementations will work, but only standard mode Windows will be available, because DesqView and Windows enhanced mode cannot be run concurrently.

Other implementors are invited to provide copies of their implementations to be run through the same tests, the results to be reported in a future issue of *TUGboat*. I am willing to accept hints and suggestions from the implementors about how to make the tests run as efficiently as possible. I am also willing to send out any files which cause problems and rerun the tests after the problems have been solved.

This test would not be what it is without valuable advice and some test files from Barbara Beeton.

◇ Erich Neuwirth
Institute for Statistics and
Computer Science
University of Vienna
Universitätsstraße 5/9
A-1010 Vienna, Austria
Internet: a4422dab@VM.UniVie.AC.AT

Warnings

Solution to the puzzle from *TUGboat* 13#2: Where does this character come from?

Frank Mittelbach

Puzzle:

> If some complex macro defined by you produces funny extra characters like "Ω" or "œ" in the output, what kind of mistake could be the reason?

After the above puzzle was published several people sent me their solutions and they all thought of problems that I didn't have in mind when I was writing this short article.

Indeed, it is possible to sometimes get a weird character at the end of an input file, namely 'æ'. This is a control-Z (`^Z`), the old end-of-file marker from the DOS operating system. This character once marked the end of a file but is obsolete with newer versions of DOS. Nevertheless, many editors and other programs still write such a mark at the end of a file, and when such a file gets transferred to another operating system the character suddenly becomes an ordinary source character, namely the character in position 26 of the current font. You can try this, by typing `^^Z` in a document (the double hat is TEX's notation for a control character). In principle, any character may show up in your document due to incorrect transfer protocols between different operating systems, but the 'æ' is probably the most common one. In an earlier *TUGboat* Barbara Beeton discussed the sad story of `^^M` behaving differently on different TEX installations due to operating system differences [BB88].

But I wasn't thinking about file transfer problems. I was talking about macro definitions that all of the sudden produce additional and undesired characters. So here follows my original answer:

The above riddle comes from some real life experience during the implementation of a complex macro for LATEX3. One evaluation of this macro under `\tracingall` results in more than 700 lines of trace information which made debugging this way somewhat unattractive. Eventually I found the source of these extra characters to be an innocent `\voidb@x` which I had forgotten to remove after changing parts of the code.

For those who never heard of this name, a short explanation: `\voidb@x` is a symbolic name for one of the 256 internal box registers of TEX. It is declared in the `plain.tex` format and, as the name suggests, should always be void, i.e., it's a constant. These symbolic names are declared with a function called `\newbox`⟨*box*⟩ that allocates for ⟨*box*⟩ a new unique box register that later on can be referred to via this name, e.g.,

`\setbox`⟨*box*⟩ `= \hbox{...}`

For other types of registers in TEX there exist similar functions to create symbolic names; for example, `\newcount`⟨*cnt*⟩ makes ⟨*cnt*⟩ a symbolic name referring to some unique internal integer register. But the `\newbox` command is somewhat special. The use of ⟨*box*⟩ declared with it does not mean "use the box register that I represent"; instead it means "typeset the character whose number corresponds to the box register I represent". Only when ⟨*box*⟩ is preceded by a "box" command (like `\setbox`, `\unhbox`, etc.) is it interpreted as a box register. Therefore, a ⟨*box*⟩ out of sequence silently typesets some character in the current font; in such a case it is equivalent to

`\char`⟨*no of the register* ⟨*box*⟩ *represents*⟩

This is a speciality of `\newbox`; all other types of symbolic names are always interpreted as referring to an internal register. For example, a ⟨*dimension*⟩ declared with `\newdimen` which is used out of sequence will be interpreted as an assignment to the dimension register and the following tokens are scanned for a dimension. This will usually result in an error, but if one is unlucky enough (e.g., if a dimension follows) one will get an equally weird behavior.

So far, I have explained this problem as a plain TEX problem, but actually the same might happen to a LATEX user who declares a "save-box" with `\newsavebox{`⟨*box*⟩`}` and later on uses ⟨*box*⟩ but forgets to call it via the `\usebox` command. The second weird character in the puzzle above was generated this way by saying

```
\newsavebox{\errorbox}
... or ``\errorbox'' in ...
```

From this we can deduce that `\newsavebox` allocated box register 27 for `\errorbox` since this is the font position for "œ" in the Computer Modern fonts.

References

[BB88] Barbara Beeton. Controlling <ctrl-M>. *TUGboat* 9(2):182–183 (August 1988).

◇ Frank Mittelbach
Electronic Data Systems (Deutschland) GmbH
Eisenstraße 56 (N15)
D-6090 Rüsselsheim
Federal Republic of Germany
Mittelbach@mzdmza.zdv.Uni-Mainz.de

Macros

The bag of tricks

Victor Eijkhout

G'day, my friends in TEX. Today I want to address two slightly TEXnical points. The first is the practical issue of how to format macros, the second is in response to comments from readers.

1 How to lay out a macro

If you write macros that will sooner or later be given to other people, you may want to be particularly careful about your style of writing. The way you write a macro should bring out its structure. If you ever wonder 'now, where is the closing brace for this box' in your own macro, consider it an indication that other people will have trouble reading that macro too!

Here are some guidelines, distilled from my experience, and from looking at the style of formatting in *TUGboat*. The examples are all taken from *TUGboat* with spacing and line breaking intact. Although all illustrate the general principle, their style still differs in detail.

1.1 Align for readability

Recognize when things are 'on the same level' and align them vertically. For example,

```
\def\myloop #1 {
  \ifx #1\end\else
    \myoperation #1
    \myloop
  \fi
}
```

or

```
\csname
  \ifx\a\b mid%
  \else
    \ifx\ad\d top\fi
  \fi insert%
\endcsname
```

1.2 Indent for readability

Look for 'levels' in your input, and give the next level more indentation. For instance, indent nested definitions:

```
\gdef\mathsraise
{%
    \begingroup
    \def \@ctivateall
    {%
        \count 0 = 0
        \loop
...
```

or (a particularly neat effect by using four spaces):

```
\def\pullet#1\of#2\to#3{%
    \def#3{\outofrange\}%
...
```

Indent the material in a box:

```
\def\EnvMakeBox#1#2{
 \setbox#1\vbox{
  \parindent0pt
...
```

Indent statements that have to be broken because they are too long:

```
\expandafter\expandafter
   \expandafter#1%
      \expandafter\expandafter
         \expandafter#3\expandafter#3#4
```

Do you wonder about the percent sign after #1 and the lack of one on the last line? The first one is necessary to prevent an unwanted space in the middle of a statement. There is no percent sign necessary at the end of the whole statement because the author apparently knows that this statement will be executed in a place where spaces are ignored. See the next section.

1.3 But what about all those spaces?

I have met people whose experiences with unwanted spaces were so traumatic that they didn't indent their macros any more, and didn't put spaces in them either. This is unnecessary, and sometimes even dangerous. Here are some rules.[1]

- Spaces at the start of a line are irrelevant, and
- a space or a line end after a control sequence is irrelevant. This point and the previous are true unless a macro such as \obeyspaces appeared earlier. On the other hand,
- A space or a line end after a control symbol, that is a backslash followed by anything but a letter, for instance \}, *is* relevant; this is not true for control space: \␣.
- Multiple spaces are almost always equivalent to just one space.
- Spaces are a good idea after a number, but be careful, not all numbers are numbers. For instance, you want a space (or a line end) after

[1] This is no attempt at explanation, and these rules are not 100% accurate either.

`\pageno=13`, `\box37` or `\ifodd42`, but not after `\multispan5`, `\magstep2` (`\multispan` and `\magstep` are macros with one parameter), or `#1` (neither in the definition parameter text or the body of a macro).

There is more to this number business; for instance, after control sequences that expand to numbers you sometimes want to place `\relax` analogous to the space after a genuine number. But this would be taking us too far.

1.4 When is a space not a space?

Sometimes a macro can contain spaces in the input that are relevant, but that you don't care about anyway. This happens if that macro or some part of it will occur in vertical mode. If TEX is in vertical mode it ignores spaces. Thus, if you write

```
\hbox{
 \Title
}
```

you get an unwanted space after the opening brace, but not in

```
\vbox{
 \noindent\Title
}
```

because TEX starts vertical boxes in vertical mode. (Makes sense, doesn't it?) Similarly, often you know that a part of a macro will be in vertical mode:

```
\def\Heading#1{\par
  \hbox{\bf #1}
  \nobreak}
```

After the `\par` TEX is in vertical mode, so the space after the `\hbox` is irrelevant.

In general it is advisable to put comment signs at line ends only if a space there can find its way into the output. If the space is guaranteed to disappear, as in the above examples, an extra comment sign would only confuse the readers who understand this point, whereas the readers who are (blissfully!) unaware of it will not be bothered by its absence.

(A bonus remark for advanced TEX users: most spaces in output routines are also irrelevant.)

2 Skip that question!

In a previous issue of *TUGboat* I wrote about conditionals such as

```
\def\ifUndefinedCS#1{%
  \expandafter\ifx
    \csname#1\endcsname\relax}
```

There is a problem with such home-made conditionals: TEX treats them as ordinary macros, so if you try to nest them in another conditional as

```
\ifnum ...
  \ifUndefinedCS{...}
  \else ...
  \fi
\fi
```

TEX will get confused if the outer conditional is false: it matches that conditional with the first `\else` or `\fi` that it finds.

Here's a way out, proposed by the big K himself, and used extensively in Stephan von Bechtolsheim's forthcoming book *TEX in Practice*. Define

```
\def\UndefinedCS#1{00\fi
  \expandafter\ifx
    \csname#1\endcsname\relax}
```

and use it as

```
\if\UndefinedCS{...}
```

Now TEX sees a conditional, and this conditional is matched up correctly if you nest it.

And that's it for this time. `\relax` and hang on to your towel!

◇ Victor Eijkhout
Department of Computer Science
University of Tennessee at
Knoxville
Knoxville TN 37996-1301
Internet: `eijkhout@cs.utk.edu`

Too Many Errors

Jonathan Fine

One of the attractions of TEX the program is that it is substantially without error. Sadly, the same is not always true for macros written for TEX. I would like to see a regular column in *TUGboat* where errors discovered in previous issues are reported. Perhaps a small prize could be given each year to the person who finds the most. Here is my contribution for the June 1991 issue (vol. 12, no. 2).

Some tools etc.: Part I—Lincoln Durst (p.248–252) Here, the error is slightly complicated to explain. The macro `\Pageno` (p.252) is intended to help write zero-padded page numbers (001, ..., 009, 010, ..., 999) to an index file. Its use is within a context equivalent to

```
\edef\next{\write\inx{\Pageno}}
```

`\next`

but the definition supplied by Durst (not reproduced here) will produce `0010` if called within text that is considered (i.e. typeset) for page 9 but actually appears on page 10.

This will not happen if the `\noexpand`s are taken from `\Pageno`, and `\noexpand\Pageno` used in place of `\noexpand\number\pageno` in Durst's `\makexref` macro.

The bag of tricks — Victor Eijkhout (p.260) Here, two macros `\storecat` and `\restorecat` are defined, to be used as in `file1`. (In the article, `\storecat` is mistakenly also called `\savecat`.)

```
% file1.tex
\storecat @
  % macros
  % @ is a letter
\restorecat @
```

The purpose is to allow `@` to be a letter within `file1`, but then to restore it *to what it was before*, when `file1` is finished.

But now suppose we input `file2` below, using the definitions supplied by Eijkhout.

```
\storecat @
% file2.tex
  \input file1
  % macros
\restorecat @
```

When `file2` is finished, `@` will have `\catcode 11`, *regardless of what it was before* `file2` *was input.* This is not what is wanted.

Babel, a multilingual etc. — Johannes Brahms (p.291-301) This article is inconsistent in its use of `%` to eliminate the space tokens that TEX would otherwise place after `{` and `}` characters at the end of a line.

The use of `\expandafter` in Fig. 1, and its second use in Fig. 4, add nothing and are redundant.

These defects are relatively slight, for they do not prevent the macros from working as advertised.

Letter from Victor Eijkhout (p.303) Here, he quotes a user as writing

```
\def\caption#1{\hbox{\hbox{ ... }\vtop{#1}}
```

This line has 4 left braces but only 3 right braces.

Conclusion Errors lie in the dark side of programming. Usually, nobody likes making them, finding them, reporting them, or being told about them. We all wish that they did not exist. I hope that this letter has not made me any enemies, and has encouraged authors to submit more articles with fewer errors.

◇ Jonathan Fine
203 Coldhams Lane
Cambridge
CB1 3HY
England
J.Fine@pmms.cam.ac.uk

One error less

Victor Eijkhout

Jonathan Fine correctly points out that the macros `\storecat` and `\restorecat` cannot be used nested. This problem cannot be remedied by using grouping, because these macros may enclose arbitrary pieces of code, and you may not want a group around them.

Here is a solution: the call `\storecat\%` defines the control sequence `\restorecat%` to restore the category code *and restore the previous definition of itself.*

```
\def\storecat#1{%
 \count255 =\escapechar
 \escapechar=-1\relax
 \csarg\ifx{restorecat\string#1}\relax
            \toks0{\relax}\else
     \edef\act{\toks0{%
       \CSname{restorecat\string#1}}}\act
     \toks0\xp\xp\xp{\the\toks0}%
 \fi
 \csarg\edef{restorecat\string#1}%
  {\catcode`\string#1=\the\catcode
   \expandafter`\string#1%
   \def\CSname{restorecat\string#1}%
   {\the\toks0}}%
 \catcode\expandafter`\string#1=12\relax
 \escapechar=\count255 }
```

where the auxiliary macros

```
\let\xp\expandafter
\def\csarg#1#2{%
    \expandafter#1\csname#2\endcsname}
\def\CSname#1{%
    \xp\noexpand\csname#1\endcsname}
```

are used. The definition of `\restorecat` stays the same.

Here is some test input:

```
\storecat\%
\catcode`\%=3
    \storecat\%
    \catcode`\%=4
        \storecat\%
        \catcode`\%=5
        \restorecat\%
    \showthe\catcode`\%
    \restorecat%
\showthe\catcode`\%
\restorecat\%
\showthe\catcode`\%
```

As a whimsical aside, the double assignment to `\toks0` can also be done in one statement:

```
\toks0\xp\xp\xp\xp\xp\xp\xp
   \xp\xp\xp\xp\xp\xp\xp\xp
  {\xp\xp\xp\csname\xp\string\xp
   \restorecat\string#1\endcsname}
```

using 15 consecutive `\expandafter`s.

◇ Victor Eijkhout
Department of Computer Science
University of Tennessee
104 Ayres Hall
Knoxville, Tennessee 37996, USA
eijkhout@cs.utk.edu

ZzTEX: A Macro Package for Books

Paul C. Anagnostopoulos

Introduction

ZzTEX is a macro package for producing books, journals, and technical documentation. The primary advantage of ZzTEX is its design flexibility, which makes it well-suited to typesetting books according to the specifications of a professional book designer. During the past three years, I and my associates have used the package to produce approximately 25 books, ranging from the 100-page journal, *System Dynamics Review*, published by John Wiley & Sons, to the 1400-page book, *VMS Internals and Data Structures, Version 5.2*, published by Digital Press. In this article I hope to give you a taste of some of ZzTEX's more interesting technical aspects. Future articles will delve deeper into the details of the macros themselves.

I was initiated into the composition and typesetting business when I agreed to compose my own book for Digital Press. I had written the book using LATEX, and continued to use it for the composition. As a software engineer, I found it impossible not to fall in love with book production: finally, an endeavor that produces a concrete work of art as its end product, rather than some ethereal software. However, I needed more design and page-makeup flexibility than LATEX had to offer, so I undertook to write my own macro package, which I subsequently named ZzTEX, after a rock group from Texas. Design flexibility is of paramount importance when producing books according to typographers' designs; neither they nor the publishers like to hear "I'm sorry, that element is too difficult to typeset."

The first book produced with ZzTEX was a real struggle. It took about two weeks to create the design file and produce sample pages. With time, I refined the package and added many new features. Each enhancement was a direct result of a design requirement in a book I had produced, so I believe ZzTEX is a practical, realistic macro package. Furthermore, my knowledge of the publishing business grew with each book. Now when I receive a design specification I can produce sample pages in less than a day. The package includes approximately 7,200 lines of TEX code and various utilities written in AWK.

The ZzTEX macro package and manual are available from the author for a nominal fee. The macro package may be freely distributed, but the manual is copyrighted and must be ordered from the author. ZzTEX uses significant amounts of

TEX memory, so the author recommends a TEX implementation that has memory areas at least doubled in size.

The Block

In general, each of ZzTEX's typographic elements is assembled from a fundamental construct called the *block*. A block "contains" the text of the element, and separates its text from that of surrounding blocks. Here is an example of a block that produces a bulleted list:

```
\list{bullet}
\item This is the first item of the
bulleted list.
\item This is the second item of
the list.  It can contain many
lines of text, and even multiple
paragraphs.
\item This is the last item.
\endlist
```

The `\list` command begins a list block. The argument in braces is called the *block type*, which allows you to specify arbitrarily many list designs. The text of the list begins after the `\list` command and ends at the `\endlist` command. Each item in the list is initiated with the `\item` command.

The list block implementation in the ZzTEX macro package provides a "generic list formatter" that has the capability to produce almost any style of list. In order to format a particular type of list for a particular book design, the block accepts a set of *design parameters* that directs its formatting of that list (e.g., the `\leftindent` parameter determines the indentation of the list items). You are responsible for providing the design parameters for each type of list, placing them in your book's *design file* (see next section). The design file contains a *design macro* for each type of block element in the book; the design macro encloses the specifications of the block's design parameters.

ZzTEX provides approximately 35 kinds of blocks, many of which accept a type argument to allow an unlimited number of variations.

The power of the block lies in the steps that ZzTEX takes when it begins and ends a block. When a block is started, ZzTEX performs the following steps:

1. Automatically closes any blocks that are terminated by the new block. For example, the section block terminates a preceding section block.
2. Opens the block scope by starting a group. This hides any parameter changes made inside the block, allowing parameters to revert to their previous values when the block terminates.
3. Stores the `\baselineskip`, `\parskip`, and `\parindent` of the enclosing block in three special parameters, thus making the surrounding values available within the new block (after all, these parameters may be changed within the block).
4. Increments a block-specific depth counter.
5. Invokes the design macro for the block. Design macros are contained in the design file loaded by ZzTEX at the beginning of the run (see next section). In the case of lists, there is a separate design macro for each type of list. The design macro establishes a set of parameters that controls further processing of the block.
6. Increments a block-specific sequence counter. If appropriate, this counter can be used to number the instances of the block, as might be done with sections or footnotes.
7. Resets the sequence counter of any subordinate blocks. ZzTEX assumes the existence of certain block relationships, such as the standard section, subsection, and subsubsection hierarchy, and resets counters accordingly. Furthermore, ZzTEX provides the `\resetnumber` design parameter on most blocks, with which you can explicitly specify other sequence counters to be reset.
8. Invokes a command (defined in the design macro) that formats the *composite number* for the block. For example, the composite number for a subsection might be '3.2.5', for a table '3-8'.
9. Performs any formatting required at the beginning of the block, including vertical space above and perhaps a title.

Once the contents of the block are typeset and ZzTEX encounters the ending command, it performs these steps:

1. Checks to make sure that the ending command had a matching starting command.
2. Recursively closes any subordinate blocks that are still open (e.g., the `\endsection` command automatically closes any subsection in progress).
3. Performs any formatting required at the end of the block, including vertical space below.
4. Decrements the depth counter.
5. Closes the block scope, discarding any parameter changes made in the block.

The Design File

A book's design specification is embodied in the ZzTEX design file, which includes a design macro for each element in the book. The design macro for a particular element specifies values for various *design parameters* that determine the formatting of the element and control behind-the-scenes activities such as the generation of marks. In addition, the design file contains commands that establish the *font table*, a matrix that correlates type sizes and styles. The design file for a typical book might comprise 50 design macros and 200 font table commands, with a total of 800 lines.

Most of the elements in a book are realized in ZzTEX with a block. Your design file contains the design macros for all the blocks used in your book. There are no block designs embedded in ZzTEX, only generalized macros that can format a block given your design parameters. Therefore, as far as design is concerned, ZzTEX is a tabula rasa waiting for your book's design. You must always include a design macro for a special block called the *document block*. The document block design includes parameters that specify the overall design of the book, plus parameters that control the look of the main text paragraphs (e.g., their paragraph skip and indentation). Figure 1 illustrates the document block design macro for a book I recently completed. In addition, the design file might contain design macros for bulleted lists, numbered lists, and plain lists. It might contain a macro for computer program examples in running text, and another for program examples in figures. And it might contain macros for chapters, appendixes, sections, and subsections.

Figure 2 illustrates the bulleted list design macros from the same book. The name of the main list macro is `\listbulletidesign`: '`list`' is the name of the block, '`bullet`' is the block type, and '`i`' is the depth. Similarly, the name of the sublist macro is `\listbulletiidesign`. This naming scheme accommodates many types, or flavors, of the same block, and also different designs for first-, second-, and third-level nested lists. The first thing the sublist macro does is invoke the main list macro; this establishes all of the main list design parameters. Then the sublist macro redefines only those parameters that are different in the sublist design. Design macros can be arbitrarily interdefined in this manner. Another common reason for defining one block in terms of another occurs when you want a design for numbered lists. The first-level numbered list macro

```
\def \documentdesign {%
  \setflag\cropmarks = \true
  \eveninnermargin = 1in
  \evenlefttextmargin = 4pc
  \evenrighttextmargin = 0pt
  \footerheight = 26pt
  \headerheight = 17.5pt
  \headmargin = .5625in
  \hoffset = -.25in
  \maxbottomcolumnfloats = 3
  \maxtopcolumnfloats = 3
  \oddinnermargin = .646in
  \oddlefttextmargin = 4pc
  \oddrighttextmargin = 0pt
  \parindent = 10pt
  \parskip = 0pt
  \setflag\PostScriptoutput = \true
  \textareaheight = 526.5pt
  \textareawidth = 28pc
  \topskip = 13.5pt
  \trimheight = 9.25in
  \trimwidth = 7in
  \voffset = -.125in}
```

Figure 1

can invoke the first-level bulleted list macro, thereby sharing common parameters such as `\aboveskip`, `\belowskip`, and `\bodyfont`.

Careful page composition often requires that individual blocks be adjusted. The `\with` command is used as a prefix on a block command to alter one or more design parameters for that particular instance of the block. This is how you can change the space above and below a list:

```
\with{\aboveskip=18pt plus 1.2pt
      \belowskip=\aboveskip}
\list{bullet}
```

ZzTEX performs `\with` assignments *after* it invokes the block's design macro.

The Font Table

One of my primary goals in creating ZzTEX was to allow complete flexibility in selecting fonts. In fact, I have never typeset a book using Computer Modern Roman. (I have used Computer Modern Typewriter, because many book designers realize it is better than other available monospaced fonts.) ZzTEX employs a *font table* to select fonts. You can think of the font table as a matrix with rows corresponding to type sizes and columns to type styles. Figure 3 illustrates a simple font table.

Three steps are required to set up the font table:

```
\def \listbulletidesign {%
  \aboveskip = 21pt plus 1.6pt
                    minus 3.1pt
  \belowskip = \aboveskip
  \bodyfont = {}% Same as surrounding.
  \interitemskip = 15pt
  \def \labelformat ##1{##1\hfil}%
  \labelshift = -\enclosingparindent
  \def \labeltext {%
    \centeronxheight{\bul .}}%
  \labelwidth = \enclosingparindent
  \leftindent = \labelwidth
  \parindent = 0pt
  \parskip = 6pt
  \rightindent = 0pt
  \width = \naturalwidth}

\def \listbulletiidesign {%
  \listbulletidesign
  \aboveskip = 18.25pt plus 1pt
                       minus 2.7pt
  \belowskip = \aboveskip
  \labelshift = -8pt
  \def \labeltext {--}%
  \labelwidth = 8pt
  \leftindent = \labelwidth}
```

Figure 2

	Type Style		
Type size	\rm	\it	\dbf
\chapsize	–	–	24' Optima Bold
\aheadsize	–	–	12' Optima Bold
\textsize	10' Nofret Regular	10' Nofret Italic	–
\ftntsize	7' Nofret Regular	7' Nofret Italic	–

Figure 3

1. Define any type styles you need in addition to the built-in ones (Roman, math italic, math symbol, math extended symbol, PostScript symbol, italic, bold, bold italic, and typewriter). The definition includes a specification of the character set encoding used by the style (e.g., Roman vs. italic vs. monospace). Knowing the encoding, ZzTEX can, for example, automatically insert an italic correction after italic text.
2. Define all the fonts you need.
3. Define the logical type sizes you need (there are no built-in ones). The size definition includes the baseline-to-baseline distance. After each size definition, set the fonts for those styles that appear in that size. Styles that are used in math require three fonts (text, script, and scriptscript); other styles require one font.

```
\definetypestyle{\sb}{\encoderoman}
\definetypestyle{\sbi}{\encodeitalic}
\definetypestyle{\bul}{\encoderoman}
\definetypestyle{\drm}{\encoderoman}
\definetypestyle{\dbf}{\encoderoman}

\definefont{\twentyfourdrm}{ss at 24pt}
\definefont{\twentyfourdbf}{ssb at 24pt}
\definefont{\eighteendrm}{ss at 18pt}
\definefont{\ninetqtt}{cmtt10 at 9.75pt}
\definefont{\nineqrm}{sr at 9.25pt}
\definefont{\nineqmit}{cmmi10 at 9.25pt}
\definefont{\nineqmsy}{cmsy10 at 9.25pt}
\definefont{\nineqmex}{cmex10 at 9.25pt}
\definefont{\nineqit}{sri at 9.25pt}
\definefont{\nineqsb}{srsb at 9.25pt}
\definefont{\nineqsbi}{srsbi at 9.25pt}
\definefont{\sevenrm}{sr at 7pt}
\definefont{\sevenmit}{cmmi10 at 7pt}
\definefont{\sevenmsy}{cmsy10 at 7pt}
\definefont{\sevenit}{sri at 7pt}

\definetypesize{\chaptersize}{24/28}
\setfont{\chaptersize}{\drm}
        {\twentyfourdrm}
\setfont{\chaptersize}{\dbf}
        {\twentyfourdbf}

\definetypesize{\textsize}{9.25/13.5}
\setfontmath{\textsize}{\rm}{\nineqrm}
            {\sevenrm}{\fiverm}
\setfontmath{\textsize}{\mit}{\nineqmit}
            {\sevenmit}{\fivemit}
\setfontmath{\textsize}{\msy}{\nineqmsy}
            {\sevenmsy}{\fivemsy}
\setfontmath{\textsize}{\mex}{\nineqmex}
            {\nineqmex}{\nineqmex}
\setfontmath{\textsize}{\it}{\nineqit}
            {\sevenit}{\fiveit}
\setfont{\textsize}{\tt}{\ninetqtt}
\setfont{\textsize}{\sb}{\nineqsb}
\setfont{\textsize}{\sbi}{\nineqsbi}
\setfont{\textsize}{\bul}{\eighteendrm}
```

Figure 4

Figure 4 shows a portion of the font table for a book.

```
\input zztex

\selectdesign{merusi}

\document
\copyident{\sevenrm Merusi first pages}
\printart{\true}

\setdivisions{chap1,chap2}

\division{front}
\setpagenumber{1}
\setfoliostyle{\decimal}
\division{chap1}
\division{chap2}
\division{chap3}
\division{chap4}
\division{chap5}
\division{appa}
\division{appb}
\division{index}

\enddocument
```

Figure 5

In addition to the main font table, ZzTEX provides a second table that specifies *style relationships*. There is a built-in relationship called `\emph` that you use to produce emphasis. The relationship table specifies that Roman is emphasized with italic, and vice versa. Furthermore, bold is emphasized with bold italic, and vice versa. You can add additional entries for emphasis, and invent your own relationships such as `\smallcaps`, `\newterm`, or `\varname`.

The Division

Any book longer than about 20 pages is best broken into *divisions*, each of which typically contains the material in one chapter. The entire book is represented by a *root file*, and the root file incorporates each division with a `\division` command. Figure 5 shows a small root file. You can use the `\setdivisions` command to select specific divisions for processing.

Associated with the root file and each division file is a *division cross-reference file* that contains the following information:

- An entry for each title that should appear in a table of contents, list of figures, list of tables, or a similar listing of other floating elements. ZzTEX allows you to define additional types of floating elements (e.g., computer program code) and produce a listing of those elements.
- An entry for each symbolic tag that is referred to elsewhere in the book.
- An entry for each endnote.
- A single *snapshot* at the end. The snapshot contains the division's final page number, chapter number, section number, and so forth.

At the end of a run, ZzTEX combines all the root and division cross-reference files into one *composite cross-reference file*. The composite file is considered the master list of cross-reference information.

At the beginning of a run, ZzTEX:

1. Loads the composite cross-reference file to obtain the symbolic cross-reference tags. (All tags in a book must be unique, so if you are producing a book from multiple independent authors, you may have to alter tags during conversion to make them unique.)
2. Starts a cross-reference file for the root file.

For each division, ZzTEX:

1. Adds an entry to the root cross-reference file that names the division cross-reference file.
2. Creates the division cross-reference file.
3. Writes table of contents entries, cross-reference tags, and endnotes to the division file.
4. Finishes the division file with the snapshot of that division.

At the end of a *successful* run, ZzTEX:

1. Closes the root cross-reference file.
2. Combines the root and division cross-reference files into one composite cross-reference file.

Whenever ZzTEX needs cross-reference information, it consults the composite cross-reference file. Thus, when you send a book to someone else for processing, you send only the TEX source files and the single composite file. If TEX encounters an error processing your book, and you terminate the run, that division's cross-reference file is invalid, but the composite file still accurately reflects the previous run. So when you reprocess the book to correct the error, the cross-reference information is still intact. The second reprocessing run behaves just like the first run.

If the `\setdivisions` command excludes a division, ZzTEX does not process it. Instead, it searches the composite cross-reference file for the division's snapshot and updates important counters such as the page number and chapter number so they reflect the state of affairs *at the end* of the division. In this way, the next division processed will appear to be in the correct place in the book.

ZzTEX writes "moving arguments", such as titles, into the cross-reference files without expansion, so that no "protect" mechanism is needed. A special command, `\adjusttitle`, allows you to customize the format of a title where it appears in the main text, a table of contents, a running head, or a textual cross-reference. An adjustment can be as simple as a line break or as complicated as a footnote.

Vertical Spacing

One of the most difficult tasks I encountered in creating ZzTEX was to ensure consistent vertical space between elements. If the book designer requests 24 points base-to-base above an A-head and 16 points below, then ZzTEX must produce that much space, regardless of the element above the A-head, the size of the A-head, or whether there is text or a B-head immediately below it. Allowing some stretch and shrink above a heading does not diminish the need for consistent nominal space. I solved the problem with the *vertical spacing environment* (not to be confused with a LATEX environment).

ZzTEX provides six commands to produce vertical space:

- `\bbskipabove`. This command specifies a certain base-to-base space between the previous element and the next. If two or more of these commands appear in a row, the space from the *first* one prevails.
- `\bbskipbelow`. This command specifies a certain base-to-base space between the previous element and the next. This command is not used to produce vertical space at the end of a block (see next item). If two or more of these commands appear in a row, the space from the *last* one prevails.
- `\bbskipbelowblock`. This command is equivalent to `\bbskipbelow`, but must be used to produce vertical space at the end of a block. It compensates for any change in the `\baselineskip` and `\parskip` values that might occur after the block terminates.
- `\bbskipbelowblockpar`. This command performs the same functions as `\bbskipbelowblock`, but also checks whether the next element begins a new paragraph. If not, it ensures that the paragraph continuation is not indented.
- `\vsink`. This command specifies a certain base-to-base distance between the top of the type page and the next element. You can use it more than once on a single page, usually in front matter to format the half title or full title page.
- `\vspace`. This command adds additional vertical space between two elements that is independent of any other explicit or implicit base-to-base space between those elements. Thus it replaces `\vskip`.

The first four commands also accept an argument that specifies the page-break penalty inserted above the vertical space. If a `\bbskipabove` command follows one of the `\bbskipbelow` commands, the maximum of the two spaces is used. If the reverse occurs, an error is signaled.

In order for vertical space to be consistent, you must use these six commands wherever you request explicit vertical space. However, most vertical space is produced implicitly at the beginning and end of a block. Any block that produces vertical space accepts the `\aboveskip` and `\belowskip` design parameters, which specify the space above and below the block, respectively. The macros that implement the block use `\bbskipabove` to produce the space above the block, and `\bbskipbelowblock` or `\bbskipbelowblockpar` to produce the space below. Therefore, you usually only need `\vsink` to format pages such as title pages, or `\vspace` to force more or less space between certain elements for aesthetic purposes. You occasionally have to use `\bbskipabove` or `\bbskipbelow` to obtain the correct base-to-base distance between two elements that are not blocks (e.g., a title and subtitle).

The vertical spacing environment is maintained as a stack of structures, for reasons explained below. The structure at each level of the stack includes the following data items:

- The type of the previous vertical space: none; inter-paragraph space specified by `\parskip`; space above, produced by `\bbskipabove`; or space below, produced by `\bbskipbelow` and friends.
- The page-break penalty associated with the previous vertical space.
- The base-to-base space requested for the previous vertical space.
- The actual glue ZzTEX inserted for the previous vertical space.

By carefully inspecting and maintaining these items, the vertical spacing commands consistently produce the correct amount of space. The environment is on a stack because vertical spacing within some blocks, such as floating figures, is independent of the vertical spacing in progress in the surrounding text. When a

floating figure begins, it pushes the current vertical spacing environment on the stack, and reinitializes the environment. The vertical spacing within the figure thus begins afresh, unaffected by the space above the figure. When the figure block terminates, it pops the stack. Each level of the stack is simply implemented as a numbered definition containing the saved values of the environment items. The items for the top level are in global variables.

The Index

The only difficulty about typesetting an index from a prepared file of entries is producing carry-over lines when a first- or second-level entry continues onto a new page. However, creating that file of entries from indexing commands in the book is a challenge. I am rarely asked to do it, because most authors do not index their books, and many publishers prefer not to use authors' indexes. Nevertheless, to support those publishers and authors who want to generate an index automatically, I developed facilities to produce an index entry file from commands in the text.

To produce an index you must first define the required *index locators*. An index locator is a particular item of information associated with an entry. The most common index locator is a page number or page range. Another common locator is the "see also" locator that refers to another index entry. When you define a locator, you specify the following information:

- The name of the locator.
- A set of attributes. The `\page` attribute specifies that a page or range of pages is associated with the locator. The `\text` attribute specifies that arbitrary text is associated with it (e.g., another index heading).
- The sorting precedence of the locator. For example, "see also" locators have a lower precedence than page locators, so they appear after the page numbers in an entry.
- The prefix text. The prefix text for a "see also" entry in English is "*See also*".
- A template that specifies how to format a single page number or the text of the locator.
- A template that specifies how to format a page range.

ZzTEX predefines the following locators, in order of precedence: the null locator (for entries without any locators), the "see" locator, the page number/range locator, a locator for each type of floating object (for referring to tables, figures, etc.), the "see also" locator, and the "consult" locator (for referring to other books).

A locator definition creates one or more macros you can use throughout a book to produce index entries. If the locator is named `command` and has the `\page` attribute, then `\xcommand` produces a locator with the current page number. The `\xcommandbegin` and `\xcommandend` macros produce locators that begin and end a page range, respectively. Each macro accepts one, two, or three arguments designating up to three levels of index headings.

To prepare an index entry file, you include the `\prepareindex` command in the root file. This command specifies an index type, thereby allowing you to have more than one index in a book (e.g., a main index, an index of commands, and an index of variables). It also specifies the name of a *index preparation file* that receives all of the control information necessary to prepare an index: root file name, index type, list of locators included in this index, and the definitions of those locators. The `\prepareindex` command also activates indexing so that ZzTEX attends to indexing commands and writes the entries into the raw index files. Without a `\prepareindex` command, index entries in source files are ignored. There is one raw index file for the root and one file for each division (as with cross-reference files).

After a ZzTEX run, three steps are required to produce the index described by one particular index preparation file:

1. An AWK program consolidates all the locators in the raw index files into one composite index file. Prefixed to each record is a six-part key that includes the headings (canonicalized for sorting), precedence, segment number, and page number. The segment number is used to separate front matter pages from main body pages. This step discards any locators that should not be included in the index.
2. The composite index file is sorted.
3. Another AWK program processes the sorted index file and creates the TEX source file for the index. All of the locators for one entry are merged into a paragraph, using the prefix text, single-page template, and page-range template specified with each locator definition. The resulting file is suitable for inclusion in an `\index` block anywhere in the book.

Conversion To and From ZzTEX

My colleagues and I have typeset books from authors who used many different document preparation

systems, including Microsoft Word, troff, VAX DOCUMENT, DECwrite, TEX, and LATEX. Not only are there many document systems, but the degree to which an author uses the features of any given system varies greatly, as does the author's level of consistency. To facilitate the conversion of these books to ZzTEX, I developed a table-driven translator called ZzTran. Because ZzTran is based on AWK, you can specify regular expression patterns that match tags in the source language. For each pattern you specify how to generate the corresponding tag in the target language (usually ZzTEX), so that a translation file consists of a table of patterns and their replacements, along with auxiliary AWK functions you write to help with the more complicated tag translations.

To translate files, ZzTran runs an AWK program that "compiles" the translation table into a pure AWK program. The compiler incorporates both driver functions that control the translation process and a standard library of utility functions (e.g., `replace()` to replace a tag, `keep()` to maintain a tag as is). ZzTran then runs the resulting AWK program, which reads a source file and produces the corresponding target file. ZzTran can usually accomplish about 95% of the translation automatically, so that very little must be done by hand.

Occasionally, after typesetting, I am asked to translate the final ZzTEX files back to the author's original document preparation system. This is usually simple because the tags in ZzTEX files are more specific and complete than those in files acceptable to the author's system. The reverse translation is a matter of converting some ZzTEX tags to the original coding system and discarding the rest.

Production Methodology

A commercial book is created by a team of people. When my colleagues and I produce a book with ZzTEX we usually break the work down as follows. I am responsible for converting the author's files to ZzTEX, because this often involves some programming with ZzTran. I am also responsible for creating the design file according to the book designer's specifications, and producing sample pages that illustrate the design. Once the design is accepted by the publisher and author, I deliver the design file and the ZzTEX source files to a person who will be responsible for the composition of the book. The compositor enters copyediting changes, produces a rough set of first pages (galleys), enters proofreading changes, produces an almost-complete set of the second pages, and produces the final pages for reproduction proof or film.

If you are writing and producing your own book, you may be responsible for all of the steps outlined above. In either case, care is required to keep the master files up-to-date, and to ensure that the notations on paper manuscripts marked by copyeditors and proofreaders are incorporated back into the files completely and accurately. The biggest dilemma concerns automatic indexing: how can the compositor work on the files while the indexer "owns" them to enter index entries? In a confined computer environment, some type of source control system can be used. But, in the real world of far-and-wide subcontractors, I have not devised an acceptable solution. We usually have the indexer put the entries in a text file, convert it to ZzTEX, and typeset it independently of the rest of the book.

Future Work

Although I have been developing ZzTEX for three years, there is still significant work to be done. My to-do list includes the following:

- The multicolumn support must be redesigned. At present, ZzTEX produces multicolumn pages by treating each column as a separate logical page. This allows single-column footnotes and floating figures, but makes it difficult to balance the columns on the final page, particularly when switching back to single column format on the same page. A hybrid scheme is required, where each column is a separate logical page, but, when necessary, all columns can be collected and treated as one page.
- While the current scheme for horizontal placement of elements is flexible, its use is not intuitive. It is particularly difficult to produce atypical kinds of centering, such as centering text in an area other than the normal text measure. I want to implement a new technique involving four parameters: `\hshift`, to control horizontal shift; `\measure`, to control the text measure; `\flush`, to control whether text is flush left, flush right, or centered; and `\width`, to specify the width on which the flushing is performed.
- I want to complete the work necessary to make ZzTEX independent of Plain TEX, yet allow them to coexist. You will be able to load ZzTEX into TEX with or without the Plain TEX macros.

- The ZzTEX manual is incomplete. It is currently about 220 pages, and will expand to approximately 400 pages.

No software is ever finished, and I will continue to enhance ZzTEX. Nonetheless, I know that Donald Knuth's TEX is capable of producing truly beautiful books.

◇ Paul C. Anagnostopoulos
Windfall Software
433 Rutland Street
Carlisle, MA 01741
greek@genome.wi.edu

The \noname macros — a technical report

Jonathan Fine

Abstract

The \noname package provides a powerful environment for writing TEX macros. Its use makes macros easier to read, easier to write, and easier to document. It allows ready access to powerful control macros. It allows diagnostic and other code to be tagged for conditional inclusion. The \noname package is fully compatible with existing macros.

Here are two major features. It allows easy access to arbitary character tokens. Lines that do not begin with a white space character are comments, and are ignored.

The intention has been to provide the productive features that users of other programming languages take for granted. This article provides an outline of the history, design and implementation of the \noname package.

Acknowledgements

I would like to thank Nelson Beebe, and particularly Michael Downes, for their careful comments on an earlier version of this article.

1 Introduction

The \noname package grew out of work the author was doing two years ago. The goal was to write macros, for setting verbatim code, that would set source in a \tt font, and comments in a proportional font. This effect was to be achieved without additional mark up of the input file. Other refinements over the usual verbatim listing for source code were also desired.

In the course of this programming, extensive access to characters with \catcodes other than those usually given was desired. This proved to be a stumbling block for this project, which still awaits completion. Various programming tricks were introduced. The result of systematically developing these devices is the \noname macros. Although they have now reached the stage of being useful, there are developments being considered that will further increase their power and usefulness.

The physical activity of erecting a building commences with the digging of a hole, that will become the foundations that support the planned structure. The larger the building, the deeper the hole. The \noname package is intended to provide secure foundations for large scale collections of TEX macros.

2 Examples

Here is a line of code from plain.tex. It supports the \newif construction. It creates a control sequence \if@ that must be followed by *other* characters 'i' and 'f' with catcode *other*, i.e. 12. Such funny letters arise from use of \string.

```
{\uccode`1=`i \uccode`2=`f
  \uppercase{\gdef\if@12{}}}
```

(The purpose of \if@ is to extract the string foo from \iffoo, which is then used to construct \foofalse and \footrue. The macro \if@ is also intended to give an error if the argument to \newif is *not* of the form \if...).

Here is the same macro defined using \noname.

```
\def \if@ 'i'f {}
```

The right quote symbol '`' is an *escape character* that serves to produce a character with catcode *other*, whose character code is given by the following alphabetic constant.

Here is another example. The \noname macro definition

```
\def\spaces{ ~~~~~ }
```

defines \spaces to be a macro whose replacement text is five ordinary space tokens. (Ordinarily, special tricks are required to get a space after a control word or another space). Finally,

```
\def !\^^M { \par }
```

will in \noname define active carriage return to expand to \par.

3 Influences

This section attempts to list the various sources for the design of the \noname macros.

3.1 Knuth's WEB system. WEB allows source and documentation to be mixed in the same file, and in a disciplined manner. It implements literate programming. The WEB system allows a module to be incrementally coded. This means, to use an example of Knuth, that with a module named ⟨*Global variables in the outer block*⟩ one can add to this module as, where, and when new global variables are required. Without this feature the global variables would have to be declared all together and at once in the PASCAL source file. Such a feature is not provided, nor planned, for \noname.

Perhaps the most important idea borrowed from WEB is the introduction of a preprocessor to augment the facilities of the language.

Because Pascal is not well adapted to storing character strings, Knuth implemented a "string pool" feature. Such a device might be welcome when writing TEX macros, for storing error and other messages, which otherwise would consume large amounts of main memory. These messages could be stored either in in TEX's string pool, or in separate files on disc, to be read in as needed.

3.2 The *C* programming language. This language is used widely for both system and application programming, and its syntactic style is widely known and imitated. Source written using \noname tends to have a *C*-like appearance.

The *C* language provides a preprocessor for source files, just as does WEB. The \noname package also includes a source file preprocessor. In particular, the hash command syntax has been copied from *C*, although the functioning will be different. Also copied is the use of the colon ':' as a label, and names for some of the control macros. (The MS-DOS batch language also uses ':' as a label).

3.3 Mittelbach's doc option for LaTeX. This work showed to me that documenting source files for TEX macros was a problem. Although his solution is a significant advance, it has limitations. His open recognition

> [t]he method of documentation of TEX macros which I have introduced here should ... only be taken as a first sketch.
>
> *TUGboat* 10, no. 2 (1989), page 246

of this encouraged me to find my own solution.

The convention — that the initial character determines whether a source file line is a comment or code or a hash command — came from a wish to have a natural input scheme for the typesetting of source files.

Although the doc option gives a pleasant appearance to the large comment blocks, it leaves the macro language of TEX unchanged, and sets small comments within the code lines in a typewriter font.

WEB consists of TANGLE, which provides a language enhancement to PASCAL, and WEAVE, which typesets source files. The doc system as published provided no language extension. It leave the TEX macro language unchanged. (The later docstrip feature does, however, allow for code to be tagged to conditional inclusion).

WEAVE recognises the key words and symbols of PASCAL, and uses this when the source file is typeset, to improve the typographic quality. When doc typesets a source file it sets code lines verbatim. Apart from recognizing and indexing control words, it has no understanding of the TEX macro language. (The \noname package, by counting braces, is able to guess when a \def-inition has come to an end).

As mentioned earlier, much of the motive for \noname came from a wish to code something superior to the doc option. This turns out to be a larger project than I imagined. The \noname macros provide part of the foundation.

3.4 Smalltalk. Certain concepts in Smalltalk have had an influence on the internal coding of \noname. The idea of compiling source into an intermediate form came from Smalltalk, which uses *bytecode instructions*. It also provides an excellent example of a productive integrated programming environment.

3.5 Wirth's Modula-2. When the module concept is added to \noname, it is quite possible that the syntax, and some details of the implementation, will draw upon Modula-2.

4 Design and Implementation

4.1 The Basic Problem. A TEX macro consists of a string of tokens. A token is either a character token — i.e. a character/catcode pair — or a control sequence. The problem of producing any given macro therefore reduces to producing any given control sequence, and any given character token. The control sequence problem shall be put to one side, for except when control sequence names are used to store textual information, it is enough that the control sequence have a comprehensible name formed from a fixed collection of characters.

Arbitary character tokens are produced via careful use of the `\uppercase` command. They are placed into macros by use of the `\aftergroup` primitive. *The TEXbook*'s "dirty trick" construction of a macro whose replacement text is `\n` asterisks (p373–4) illustrates the basic technique.

4.2 `\load`, `\comp`, `\online` and `\hsl`. The macro writer will specify, using a syntax to be described later, a sequence of control sequences and character tokens, to be formed into a macro. The `\noname` macros will read and act upon this input stream — there are examples above — so as to construct the desired macro.

The `\load` command will read these macro-building instructions from a file, and place the result into TEX's memory. The `\comp` command will write the content of these instructions to a file on disc, in a form that can then be re-processed at high speed, by use of the `\hsl` command. Finally, the `\online` command is like `\load`, except that it takes its input from the console, rather than a file.

5 Structure of source files

5.1 White space. All white space is ignored (unless preceded by a `\noname` escape character). It is no longer necessary to use '`%`' symbols to prevent space tokens creeping into your macros.

5.2 Comment lines. Any line that does not begin with white space is a comment line, and will be ignored (unless it begins with a '`#`' hash character — see below).

5.3 Escape characters. The `\noname` package has a rich range of escape characters.

- `~` produces an ordinary space token.
- `'` produces a character, with catcode *other* = 12, whose character code is given by the token immediately following "'". This token may be a white space token, or some other character, or a control symbol.
- `!` is like "'", except that it produces an *active* character.
- `|` is the *bar* construction, which allows access to arbitary character tokens. It should be followed by the `\catcode`, as a hexadecimal digit, and then the character code, as a character or a control symbol. Thus `|D` is equivalent to `!` and `|C` is equivalent to `'`.
- `:` is a label which produces an otherwise inaccessible macro, whose expansion is empty. This device is most useful when used in conjuction with the *Basic Control Macros* cited elsewhere.
- `@* ;` will produce unusual character tokens. They are intended for use with the `\CASE` and `\FIND` macros (*TUGboat*, to appear), and some other purposes.

Finally, the characters `{}$#^_&` and `%` have their usual effect.

5.4 Control words. When using `\noname`, not only can the letters `a..zA..Z` and the `@` character be used for forming control words, but also the characters `$&*_:` and the digits `0..9`. This does not interfere with the usual use of the characters, outside of control words. For example

```
\def\subscript_character{ _ }
```

defines `\subscript_character` to be a macro whose replacement text is a *subscript* character, with '`_`' as its character code.

5.5 Numeric constants. Within `plain` and LATEX the control sequence `\m@ne` is used to refer to a `\count` register whose value is fixed to be −1. This feature is provided because −1 is a ubiquitous constant. Macros run quicker, and occupy less space, if `\m@ne` is used in place of `-1`. The `\noname` package provides the same functionality, but by typing `[-1]`.

With `\noname`, the tokens `[nnn]` where `nnn` is a literal number such as `-1` or `"57` or `16` will produce a control sequence which is to store the number `nnn`. This allows the popular numeric constants to be referred to in a literal manner, rather than via cryptic names.

A similar convention applies to numeric constants specified as character constants. The character "`'`" followed by a *control symbol* such as `\x` or `\^^M` will produce a control sequence which stores a number, namely the ASCII value of `x` or `^^M` respectively.

If `[` is followed by a token that cannot begin a literal number, i.e. other than `0..9` or `+-'"`, then no special behaviour occurs. Similarly, if `'` is not followed by a control sequence, then no special behaviour occurs.

5.6 Hash commands. This is a feature borrowed from *C*. Any source line beginning with a hash `#` is a hash command. Hash commands control the processing of the file. They allow conditional inclusion of code. For example, if `\ifdebug` is `\iftrue` then the line below

```
#\ifdebug
                \checkingcode
#\fi
```

which contains `\checkingcode` will be processed, while if `\ifdebug` is `\iffalse` then this line will be skipped.

This feature allows the same file to contain several variants of the same code. Currently, LaTeX has three files — `art10.sty`, `art11.sty`, and `art12.sty` — which are identical in all aspects, except for the values of some numerical and other parameters. Use of hash commands allows these files to be described using a single source file.

This feature can also be used to maintain a single file for several versions of a macro package.

6 Control macros

The author's *Basic Control Macros* in *TUGboat* 12, no. 2 are easier to use within the `\noname` environment, for they depend on a label ':' being available. The author has also written powerful control macros `\CASE` and `\FIND`, which again depend on `\noname` features — in this case that `*` and `;` produce not ordinarily accessible character tokens.

The author is about to release a control macro `\FSA` (for Finite State Automaton). Here is an example of its use. When, on a page, one vertical item is placed beneath another, vertical space may be required, or perhaps a penalty, or some other activity. The `\FSA` device allows the decision table for such transitions to be coded in a simple, elegant, and economical manner. It will use `@*;` as delimiters.

7 Structure of `.hsl` files

The details of the fine structure of the `.hsl` files should be of little concern to macro programmers, so long as it adequate to support their needs.

Here is an extract from a `.hsl` file.

```
^^@ \FILE tutor.hsl %
^^@ \year 1992 \month 9 ...   720 %
%{ \hsl"{"c"o"n"t"r"  ...  %%%%%%%
% \tracinglostchars  ...  "{"m"a"i%
%"n"m"e"n"u"}"}}{   ...  e \online%
% \def \online"{ \e  ...  #'1"{%%%
% \immediate \write"\  ...  "{%%%%
```

Normally, during a `\hsl`, `^^@` is a comment character and `%` is ignored. This is designed to support macro library files. (In this context, see *The TeXbook*, p382-4). By setting `%` to comment, the macros can be skipped at high speed.

By setting `^^@` to ignore, it is possible to read the `\FILE` and date information in the header. The date can be used for version control and compatibility. For example, by storing multiple versions of the same macros in a single file, the most recent first, it is possible to load the macros that are in force at some given date. Simply load the first macro package whose date is before the given date. (The date is precise to time of day, in minutes, as supplied by TeX's `\time` primitive).

Currently, the optional code controlled by hash commands can be used only to generate multiple `.hsl` files, each obtained by processing a different subset of the source file. Essentially, the variant is determined at the time of the `\comp`-ile. However, the basic structure of the `.hsl` file is sufficiently rich, as to allow these variants to be combined into a *single* `.hsl` file. The setting of a flag will then control the choice of variant *at the time of the* `\hsl` *of the file.* This feature could be used to produce, if wished, a single `art.sty` file for use with LaTeX.

8 Modularity and named parameters

Many other languages restrict the scope of an identifier, so that the same identifier can be used for different purposes in different contexts. For example, in *C*, identifiers declared within a function are local to that function, while identifiers prefixed by the keyword `static` are local to the file in which they appear.

TeX has a single global name space. Consequently, each author of macros has to be sure that his or her control sequences do not clash with those of `plain`, LaTeX, or some other package. This is a burden.

Here is a related matter. In other languages the parameters to functions (the TeX equivalent is macros) are identifiers. This improves the code greatly. For example

```
\def\centerline #\text
{
  \line{ \hss \text \hss }
}
```

is easier to read and maintain.

The addition of these facilities to the `\noname` package is being investigated.

9 Performance

The single most important aspect of the performance of the `\noname` package is the degree to which it allows the macro writer to produce better code quicker. I will leave the measurement of this to others, who are able to be more objective. My experience of using `\noname` is that the code written is much easier to read after the event, and that the various helpful facilities reduce coding time by

between 10 and 30 percent. The saving will depend on the nature of the macros being written, and the extent to which they are basic. If two (or more) almost identical versions of the same file are required, then the time saving can be much greater. This is also true if active and other special characters are required.

The commands `\load` or `\online` are quick enough, on a slow machine, to process small (say 50 line) files, but become tedious for much larger files. They also have an overhead of one (1) control sequence for every new control sequence processed. This overhead will increase, and the performance fall, when the various enhancements are added. However, the `\hsl` command works at much greater speed, and with minimal overhead.

The `.hsl` files are much smaller that the source files from which they are generated. They can be combined into a single library `.hsl` file, with conditional run-time loading of the constituent parts. These features can be used to save mass storage requirement (note that file space is allocated in fixed size blocks), and reduce traffic on a network and on `email`. Note also that it can take the operating system longer to find a small macro file, than it takes TEX to process it.

However, most macros are loaded once, and then `\dump`-ed as a format file, which can then be loaded at high speed. The quality of the code will then determine the size of this file, and thus how quickly it can be loaded.

10 Future developments

Briefly, here are some projects that are under way. Much progress has been made on coding a *pretty printer* for typesetting source files written in the `\noname` dialect. It is intended that it should also be able to typeset suitably laid out *C* and *C++* source files.

A proof-of-concept prototype for a *single step debugger*, that will execute or expand TEX macros and commands one at a time, has been coded, and will also form part of the `\noname` package. I hope that it will be useful both for learning and teaching how TEX works, and also for development.

Finally, an interactive tutorial for `\noname` — consisting of TEX macros and so running within TEX — has been written.

11 Availability

This package has been developed privately. Future developments will require financial support, most likely from sale of the software.

Publishers and other major users of TEX require custom macro packages. These are either written in house by expert staff, or commissioned from outside. These packages are usually proprietary, although publishers tend to make them available when appropriate to their authors.

At the other end, there is a large mass of unsupported macro files, of variable quality, available for no cost. In addition, there are packages such as `plain`, LATEX, PICTEX.

Discussion with TEX users will reveal the technical and other merits of `\noname`, and help provide a basis for pricing, licensing, distribution and other policies.

The current version is already useful. It may take six months to add modules, named parameters and other advanced features. Also required, as in *C*, are libraries of standard functions.

A demonstration version is available (from the author only), so long as you agree to respect his intellectual property rights.

◇ Jonathan Fine
203 Coldhams Lane
Cambridge
CB1 3HY
England
J.Fine@pmms.cam.ac.uk

LaTeX

Volunteer work for the LaTeX3 project

Frank Mittelbach, Chris Rowley and Michael Downes

1 Introduction

This is a call for volunteers to help in the development of LaTeX3. There are many tasks needing to be done in support of the LaTeX3 project which can be worked on concurrently with the development of the LaTeX3 kernel. Furthermore, some tasks require special expertise not found among the core programming team. Initial research, analysis, and work on these tasks by volunteers can greatly speed up the process of integrating a number of desirable features into LaTeX3. Many of these features can be extensively developed and tested under LaTeX2.09 even before the LaTeX3 kernel is available.

Therefore we are publishing a list of tasks to the LaTeX user community through various channels and we ask readers to consider contributing some time and effort (particularly, but not exclusively, readers with expertise in the various areas touched on). The task list is distributed in the form of a LaTeX article; it is fairly readable in electronic form, and it can be printed on paper if desired.

If you are interested in working on a particular task, see Appendix A for details on how to volunteer.

The task list will be updated at regular intervals. For instructions on obtaining a copy from the public archives, see Appendix B.[1]

2 General tasks

2.1 Volunteer list management

Organization, publication and maintenance of the general volunteer task list.

List manager: George Greenwade.

2.2 Validating LaTeX2.09

Writing test files for regression testing: checking bug fixes and improvements to verify that they don't have undesirable side effects; making sure that bug fixes really correct the problem they were intended to correct; testing interaction with various document styles, style options, and environments.

We would like three kinds of validation files:

1. General documents.
2. Exhaustive tests of special environments/modules such as tables, displayed equations, theorems, floating figures, pictures, etc.
3. Bug files containing tests of all bugs that are supposed to be fixed (as well as those that are not fixed, with comments about their status).

A procedure for processing validation files has been devised; details will be furnished to anyone interested in this task.

Coordinator: Daniel Flipo
`flipo@citil.citilille.fr`

3 Syntax questions

3.1 `.sty` metacomments for smart editors

Develop conventions for documentation of styles which could be picked up by editor packages to provide editing help.

The idea is to place metacomments in `.sty` files which smart text editors (in particular) can use to get information about the 'exported' (user interface) macros for that particular style. The information would be useful for word completion and spelling checking, at least. (The auc-tex package for GNU Emacs currently has such information hard-wired for a number of common styles.) If the editor has access to the `\documentstyle` line or suitable alternative instructions it can poke about in the appropriate style files rather than using its own database.

Such information could be written out by a run with `doc.sty` on the basis of `\Describe {Macro, Env}` commands in the `.doc` file and subsequently included in the `docstrip`'ped `.sty` file. That's easy enough, but if it's to be generally useful the result ought to be somewhat standardized and in a form suitable for use by as many editors or other tools as possible.

Would conventions for supplying other information this way be useful (along the lines of the PostScript structuring conventions)?

Coordinator: David Love
JANET: `d.love@uk.ac.dl`,
BIT/INTERNET: `d.love@dl.ac.uk`

3.2 Syntax proposal for bibliographical commands

Extensions of current LaTeX syntax for `\cite` commands and bibliography commands. A number of specialties have conventions for citations and bibliographies that LaTeX 2.09 is ill equipped to handle.

[1] Editor's note: This summary is based on version 5.1 of the task list, dated 15 October 1992. The archived list contains some information not included here, such as time estimates and the names of volunteers other than the task coordinators.

David Rhead published several papers concerning the handling of bibliographies and citations [Rhe90, Rhe91a, Rhe92a, Rhe92b]. Some of them have been distributed via the `latex-l` mail list. Counter-proposals or further argumentation for David Rhead's ideas would be useful.

Coordinator: Open

3.3 Research on syntax for tables

What features are important (and not covered)? Logical representation of tabular material versus visual representation. Syntax proposal and report.

About tabular material presentation many interesting papers are published. For example, general articles [Bea86, Bea85]; LaTeX related [Car90, Car91, Rhe91b]; logical table representation [Van92]. Important work was done by Michael Spivak in [Spi89] and of course in his "Tables to die for" (T2D4). Standard books on typesetting ([But81, McL80, Chi82, Whi88], to name only a few) also usually contain important information about tabular typesetting. What is necessary is a survey of the requirements for tabular material in printing, a proposal for an extended standard syntax, and perhaps a proposal for syntax of extra features that could be provided through a separate 'super tables' module that is not loaded until the user requests it.

Coordinator: Ed Sznyter `ews@babel.com`

3.4 Research on syntax for chemistry

The typography of chemical texts is rather different from, say, mathematics. We need a taxonomist to classify the primary elements of an article or book on chemistry and suggest syntax for user commands to handle each element. What proportion of chemical diagrams can be constructed with primitive line graphics such as given by the LaTeX picture environment (with suitable extensions)? Or should diagrams just always be done in some other graphics language and imported via `\special`?

Coordinator: Chris Carruthers
`cjc@acadvm1.uottawa.ca`

3.5 Research on syntax for commutative diagrams

Commutative diagrams occur often enough in mathematical literature that even the first version of AMS-TeX back in 1981 or so included a rudimentary facility for constructing rectangular commutative diagrams. Since then several people have produced various alternatives, most involving special fonts with line segments slanted at various angles, and arrow heads. The commutative diagram macros of LAMS-TeX have arrow directions specified as vectors with the units being rows and columns rather than distances, e.g., `\arrow (1,2)` means a diagonal arrow from the current element to the element one row over and two columns down.

There is a `catmac.sty` by Michael Barr that uses the line fonts of LaTeX for drawing slanted arrows. The `XY-pic` package by Kristoffer Rose is reportedly usable with LaTeX and comes with its own line and arrowhead fonts.

For LaTeX3 we would like to see an analysis of the logical structure of commutative diagrams and recommendations on user syntax.

Coordinator: Paul Taylor `pt@doc.ic.ac.uk`

4 Research tasks

4.1 Experimenting with `\emergencystretch`

Testing the new features of TeX3 where no experience is available so far. Writing up a report.

Research on `\emergencystretch`, in particular, is an important area where the TeX community doesn't have enough experience so far, e.g., what are good values in what situations, why? What happens if... and so on. This would also make a good article for *TUGboat* if the report were given some finishing touches afterwards.

Coordinator: Open

4.2 Research on indexing commands

What kinds of indexes are needed for various fields? What kinds of indexes are needed for various specialties? What kinds of `\index` commands/syntax need to be provided for marking entries? What kinds of commands need to be provided for printing indexes after they have been processed by a program like MakeIndex?

Coordinator: Open

4.3 Research footnote/endnotes conventions

What conventions are used for various specialties? What user commands and syntax would be recommended? Report on the results.

Coordinator: Open

4.4 Syntax diagrams

Designing a command syntax (and implementation in LaTeX2.09) for syntax diagrams used to illustrate programming language syntax.

> Reference: Michael F. Plass, Charting your grammar with TeX. *TUGboat*, 2(3):39–56, November 1981.

The described syntax is probably not appropriate for LaTeX and the implementation needs refinement since it was done for TeX78 but it is a good starting point.

Coordinator: David Morgan
morgan@socs.uts.edu.au

4.5 BNF notation

Designing command syntax and prototype LaTeX-2.09 implementation for BNF (Backus-Naur) notation used to describe syntax of programming languages.

Coordinator: Mike Piff
M.Piff@sheffield.ac.uk

5 Research tasks (cont.)

5.1 Research on use of shorthand forms

In SGML there is a concept called 'short ref' which means for example that the double quote character " can be defined to produce directional quotes, blank line can be interpreted as end of paragraph, and so forth.

What kind of similar shorthand forms in ASCII files may be desirable for LaTeX users, e.g., => to be converted to ⇒, /= or <> to be converted to ≠, '? to be converted to upside-down Spanish question mark, "u to be converted to umlaut ü, and so forth. What conventions are currently in use for various kinds of documents?

Something along these lines is currently done in AMS-TeX with the @ character: @- is a shorthand meaning 'nonbreaking hyphen', @, is a shorthand meaning one-tenth of a thinspace, @> is a shorthand for an extensible right arrow, and so forth.

It is envisioned that in LaTeX3 the user will be allowed to designate certain characters to be shorthand initiator characters. For efficiency reasons, the set of allowed initial characters will probably be restricted to nonalphanumeric only.

Coordinator: Julio Sanchez jsanchez@gmv.es

5.2 Research on figures and captions

What rules are in common use for placement and formatting of floating figures and their associated captions? Propose syntax for user commands. Write report.

Placement rules for floats and their captions are so far very limited in batch formatters like TeX. We are interested in rules for such placement which are used in practice, algorithms, and possible user syntax. What could be a good user syntax for putting captions above, below, on the side, centered or top or bottom or left or right? Do we need to allow different action for different classes of floats? What do we need for multi-figure groups and their captions?

Coordinator: Sebastian Rahtz
spqr@minster.york.ac.uk

5.3 Research on the use of ^^ conventions

Check the actual use of the ^^ convention for special characters in the LaTeX community by polling as many users, organizations, mail-lists, usenet groups, etc., as possible. Write report.

In TeX the ^^ notation is sometimes used for access to unusual characters (< 32 or > 126). It would be useful to separate this function from the superscript function by assigning it to some character other than ^, if that would not be too large an inconvenience for users. One approach, for example, would be to change ^ and _ to be active characters so that they can always keep track of current math style, which would allow a better definition for \mathchoice and simplify many things having to do with math fonts. It seems that the ^^ notation is indispensable only when the character is used in a control sequence name or as a macro argument delimiter (or in hyphenation patterns?). Note: document styles are of less concern since they will have to be mostly rewritten for LaTeX3 anyway.

Coordinator: Open

5.4 Research on typographic conventions and requirements in multilingual environments

Typographic conventions differ from one language/country to another. Collect information about such conventions and try to identify the basic data-types and operations required in LaTeX3, so that most or, ideally, all features necessary for the support of many languages can be implemented in the LaTeX3 programming language.

It would be helpful also to include anything whose provision is already supported by the babel system and/or other systems: e.g., hyphenation.

Coordinator: Open

6 Miscellaneous items

6.1 Math font handling

Test math font handling in the latest release of NFSS and write up detailed comments.

Last year there was some discussion among the LaTeX3 programmers and others on how to handle math fonts under an enhanced release of NFSS for LaTeX3. The discussion finally drifted off into areas that are far beyond the scope of the LaTeX3 project but the actual questions that were raised have not

yet been answered. The only contribution that came close was the detailed suggestion and experience report by Sebastian Rahtz about the alpha release for an extended text font handling which was sent around via the `latex-l` list.

A related, but separate, subtask involves thinking about proper math font handling taking into account the papers already sent around.

Coordinator: Open

6.2 Converting numbers to textual form

Currently counter values can be displayed in certain styles, e.g., as roman numerals. But it may be interesting to extend the available commands by cardinal and ordinal representations, e.g., 5 ⟶ 'five' or 'fifth' (for example, if you wanted to refer to 'the fifth item' in a list using something like LaTeX's `\ref`). Spivak's LaMS-TeX has `\cardinal` and `\ordinal` macros to do this, for handling cross-references such as 'the fifth item in the list' where 'fifth' is supposed to be generated by a `\ref` command. The main question: How much do we need this capability? Should it be standard, or merely a nice option for those who want it? Can it be easily extended to support various language conventions? Are there other significant uses besides the cross-reference idea?

Coordinator: Open

7 Miscellaneous items (cont.)

7.1 Rewrite MakeIndex in WEB

Convert/rewrite the C source code of MakeIndex. For consistency it would seem desirable to have all auxiliary programs designed for use with LaTeX3 to be compilable in the same way as TeX. Currently this means use of the WEB language, with or without the CWEB intermediate step.

Furthermore, the MakeIndex program could use some work to deal with a few shortcomings that have become evident with the passing of time and extended usage.

Coordinator: Open

7.2 Write other auxiliary programs

Create programs for support tasks related to LaTeX documents but not part of the primary typesetting functions.

Question: what other auxiliary programs do we need? Conjectures: Compiled version of `docstrip`? Programs to help designers in creating document styles? Program for dealing with graphics files in various formats (e.g., read Bounding Box comments from a PostScript file and compute scaling and translation numbers for passing to a LaTeX `\special` command)? Checksum utility by R. Solovay for updating Nelson Beebe's standardized file headers. Auxiliary program to help in constructing complicated tables (decimal point alignment, row spanning, other fancy effects that are hard to do in TeX currently)? Auxiliary program similar to Type & Set to do interactive page-breaking/float placement?

Coordinator: Open

7.3 Bibliography style programming

Write bibliography styles for BibTeX1. The current version of BibTeX is 0.99. A reimplementation of BibTeX for LaTeX3 is under way, by Oren Patashnik. When this is finished, or perhaps even before, suitable standard bibliography styles for LaTeX3 need to be written.

Pending because of status of BibTeX1

7.4 Bibliography style requirements

Collect available BibTeX0.99 styles and, if possible, further journal and publisher requirements regarding bibliographies and analyze them. Summarize the functionality of each style, whether or not it is easily programmable with the current BibTeX, what special functions would be helpful, etc.

Coordinator: Robert Tolksdorf
`tolk@cs.tu-berlin.de`

7.5 Survey of existing LaTeX style options

Using David Jones' `TeX-Index` (and any other useful sources), evaluate the status of the many LaTeX2.09 options currently available, e.g., whether they are up-to-date, whether the authors still support them, or if unsupported, whether they are interesting enough to make it worthwhile to seek a new maintainer for them.

Write a report indicating the status of each style option, a short description of its features and, if it is not maintained, if you think it is worth upgrading or maintaining it.

`TeX-Index` is an index of (LA)TeX macros. From its documentation:

The most recent version is always available by anonymous FTP from `theory.lcs.mit.edu` in the directory `pub/tex/TeX-index`.

Copies can also be obtained from the following locations:

```
archive.cs.ruu.nl  TEX/DOC/TeX-index.Z
ftp.th-darmstadt.de  pub/tex/documentation/
  styles-and-macros.Index.Z
ftp.math.utah.edu  pub/tex/tex-index
```

Correction sheets in LaTeX

Mike Piff

Abstract

In this article the author explains how to produce minor correction sheets to a LaTeX book. The sort of corrections handled are those which involve changes to a few words or possibly rewriting a few lines on a page, but not the sort that would cause TeX to reformat the whole book because a page has stretched or shrunk too much. The author has used these macros for corrections to a book in print, but they could equally well make the last stages of proof reading and correction less irksome.

1 Introduction

Reprinting individual pages of a LaTeX book for correction can be quite a tricky task. One can always process the whole book again, but chances are that the page breaks will all be different. This can also cause cross-references and the index to be inaccurate, and so is not really suitable unless the whole book is being reset.

The following style option `corrects.sty` makes the job easier. It was developed in particular for the author's needs, but should be easy to customize.

2 The style in use

It is assumed that a root file is available to produce the book. Make a copy of this, and insert the correction style option, and also the commands

```
\correctionsonly
\renewcommand{\resetcounters}[2]{%
   \setcounter{Theorem}{#1}%
   \setcounter{Example}{#2}%
   \ignorespaces}
```

in the preamble. The former is to tell LaTeX to produce only correction sheets, and the latter is customized to the particular counters being used in the current book—in this case, theorem and example counters. We must either make a copy of the aux file from the original book, or carefully replace any `\ref` and `\pageref` commands with their expansions in the corrections file. The author's own preference was the latter; he changed his root file so that it actually contained the pages to be printed, rather than have to `\input` or `\include` copies of each chapter file and aux file. As a precaution, he also redefined `\ref` and `\pageref` so as to give an error message.

Most of the body of the book may now be discarded. In its place, we put instructions to print individual pages. At least the text of these pages must be retained. But this is where the complications begin, since it is possible to be heavily nested within several layers of environments at the start of a page, as well as being mid-paragraph in section 9 of chapter 4, and about to produce Example 4.97 and Theorem 4.25. Moreover, at the end of the page we may be in a different paragraph and a different section. The headers on the pages will have to reflect the current state, and the page must finish flush right.

None of these problems is of much significance in itself, but the combination of all of them means that we have to be pretty careful in giving the exact state of the book for each page. Some counters change rapidly from page to page, whereas some change more slowly. I have divided them accordingly, and provide explicit commands to reset the slow counters, such as `chapter` and `section`, whilst allowing the individual page instruction to reset the quicker moving sort, such as `Theorem` or `Example`.

The instructions

```
\currentstate{chapter}{4}%
   {The theory of relaxors}
\currentstate{section}{9}%
   {Covariant relaxors}
```

tell LaTeX that we are firmly in chapter 4, section 9. LaTeX now needs to know how many theorems and examples are behind us. We will tell it at the start of the next page.

But first we need to inform LaTeX of the current state of nesting of environments at the top of the next page. We can do this either by leaving the immediately preceding text in, or by just giving it a clue like this.

```
\begin{enumerate}
   \item \mbox{}
   \item \mbox{}
   \item
      \begin{itemize}
         \item\mbox{}
         %%This is the text that
         %%immediately preceded the curr-
```

The paragraph is clearly in full flow, and so we tell LaTeX to start the page flush left, where "left" means according to the level of indentation of environments.

```
\startpageflushleft{101}{24}{96}%
   -ent page.
   Now we are producing page~101.
   The next theorem will be~25.
   The next example will be~97.
```

There is also a similar `\startpage` command to produce a normally indented paragraph at the top of

the page, or to use if the first thing on the page is a theorem, say.

This page ends in mid-flow, so we let LaTeX know about this.

```
...it was clearly not too diff-%
\endpageflushright.
```

The `\endpage` command has a similar meaning, but allows normal paragraph termination in mid-line. Both cause the page to end flush bottom. Just use `\clearpage` if this page is at the end of a chapter.

It may happen that a run of two or more pages have mistakes on them. The macros in the style option are designed to make that easier to handle. At the start of a second, or subsequent, page in mid-paragraph, insert the instruction

```
\anotherpageflushleft
```

or include the command `\anotherpage` in its text if it does not have to start flush left. Terminate any such pages in the usual way. This saves having to retype the current state of the fast moving counters.

The author has found that floating figures are handled correctly, but TeX has to see the whole page where the figure is defined. Alternatively, the figure can be moved to an appropriate place on the page being printed. Clearly a large number of held-over insertions could present some problems, and the best course of action might then be to move them to the exact place where they should appear, with the "insert here" option active.

Footnotes on the current page can be handled by using the optional parameter to set the correct mark. Footnotes held over wholly from a previous page can be inserted with `\footnotetext`. A split footnote from a previous page is handled by means of the `\morefootnotetext` command, which takes the text of the footnote as its only parameter.

The final feature of this style option is the fact that *only* the pages specified above get through to the `dvi` file.

3 The style in detail

We first of all define two boolean variables. The variable `\ifcorrections` is an indicator of whether we are producing corrections or not. Its default is false, so the style will have no effect unless it is changed to true. Variable `\ifrealpage` is used internally to tell LaTeX to actually ship out the current page. Its default is true.

```
\newif\ifcorrections \correctionsfalse
\newif\ifrealpage \realpagetrue
```

To switch from this default mode, where the style has no effect, we provide a command to change the values of these two variables.

```
\def\correctionsonly{%
   \correctionstrue \realpagefalse}
```

The sectioning counters can be changed by means of the `\currentstate` command. We provide some dummy text by means of the `\mbox{}` command.

```
\def\currentstate#1#2#3{%
   \setcounter{#1}{#2}%
   \addtocounter{#1}{-1}%
   \csname #1\endcsname{#3}\mbox{}}%
```

To start a page of output, we specify the page number and then call a command `\resetcounters`, whose default meaning is to do nothing but ignore spaces. This command should be redefined in the user's preamble to take account of any counters that might need updating.

```
\def\startpage#1{\npage{#1}%
   \resetcounters}
\def\startpageflushleft#1{\npage{#1}%
   \noindent\resetcounters}
\newcommand{\resetcounters}{\ignorespaces}
```

Both commands to produce a new page make use of the `\npage` command.

```
\def\npage#1{\clearpage
   \global\realpagetrue
   \setcounter{page}{#1}}
```

The end of a page, flush right or not, is produced by the following two commands, or if appropriate by a `\clearpage`.

```
\def\endpageflushright{%
   {\parfillskip0pt\par\pagebreak}}
\def\endpage{\pagebreak}
```

If the page after this one also needs to be printed, an alternative way of producing it is to use one of the following two commands.

```
\def\anotherpageflushleft{%
   \global\realpagetrue\noindent}
\def\anotherpage{\global\realpagetrue}
```

A footnote from the previous page, part of which appears at the bottom of the current page, can be handled by inserting a left justified footnote without a mark in either the text or the footnote itself. This command should be invisible in the surrounding text, and so we make certain that it does not alter TeX' space factor.

```
\def\morefootnotetext#1{\@bsphack{%
   \def\@makefntext##1{\noindent ##1}%
   \footnotetext{#1}}\@esphack}
```

The only thing that remains is to make sure that any extraneous text, such as is created by sectioning commands, is not printed. We first take a

copy of the LaTeX command that actually ships a page to the dvi file.

```
\let\@@outputpage\@outputpage
```

The command \@outputpage is then redefined in terms of its old meaning. First, only "real" pages are printed, that is, only when \ifrealpage returns true. Then, if we are only printing correction sheets, we immediately switch off printing of pages after this one. Only the \startpage and \anotherpage commands will switch printing on again.

Note that we need to ensure that any special page style set by \thispagestyle on a page that has not been printed does not carry over to the following page. For instance, a \chapter command will generally introduce a plain page style command into the document.

The command \@outputpage should otherwise do everything that its old version did, apart from shipping the page out to the dvi file.

```
\def\@outputpage{%
   \ifrealpage\@@outputpage
   \else\global\@specialpagefalse
   \let\firstmark\botmark\fi
   \ifcorrections\global\realpagefalse\fi}
```

◇ Mike Piff
Department of Pure Mathematics
University of Sheffield
Sheffield S10 2TN
England
Janet: M.Piff@shef.ac.uk

Text merges in TeX and LaTeX

Mike Piff

Abstract

In this article the author explains how to do some standard and not so standard word processor text merges in TeX documents, using no other tools than TeX itself. A common application is to the mail merge or form letter, where names and addresses are stored in a file, together with other bits of information, and a standard letter with variable fields embedded in it is customized for every name from this file. Another application is to the pretty-printing of the contents of a database.

The macros described in textmerg.sty work equally in both plain TeX and LaTeX.

1 Introduction

It is often said that although LaTeX is good at typesetting mathematics, it is wholly unsuitable for common word processor functions such as mail merges. The latter are easy to achieve in most ordinary word processors, but in its raw state LaTeX is incapable of doing a mail merge, or, indeed, of generating the same block of text over and over again but with different parameters in each block, those parameters having been read from a subsidiary merge file. The latter file might possibly be the output from a database or any other program.

This article aims to show the reader that such a repetitive task need not be as difficult as it at first appears. In TeX, it is possible to hide many details of a facility inside a subsidiary style file, so that the user is unaware of what fearful processes are going on in the background. It is then possible to present the end-user with an extremely simple interface, perhaps simpler and more powerful than is available in other systems.

In earlier *TUGboat* articles [Bel87, Gar87, Lee86, McK87] it was shown how a standard letter could be customized by adding names and addresses from a separate file. I aim to show that it is possible to achieve far more than this with a fairly compact but general set of macros.

2 A simple example

Suppose that we have a list of student names and examination grades, one per student, and that we wish to send a letter to each student giving his/her exam grade. We must decide first what bits of information must be prepared in our subsidiary file, by looking at an example letter and finding out which items change from letter to letter.

Suppose that one instance of our letter is the following, a LaTeX example.

```
\begin{letter}{Mr Abraham L Spriggs\\
   34 Winchester Road\\
   Sheffield S99 5BX\\
   England}
   \opening{Dear Mr Spriggs,}
   This letter is to inform you
   that you obtained grade C in
   your recent examinations.
   \closing{Yours faithfully,}
\end{letter}
```

We can see that we need to know the student's title, forename(s), surname, address and grade to compose such a letter.

One of the simplest ways of achieving this effect is to prepare a file with lines of the form

```
\MyLetter{Mr}...{C}
```

for each student and then simply `\input` it into a LaTeX file in which `\MyLetter` has been defined as having five parameters. A problem with this approach is that we may not be able to coax the student database into producing such a file. Another problem is that we need something more subtle if there are fifty parameters. For example, we might want to print out the contents of the student database with one page per student, but it could be that there are fifty information fields per student. Even worse, the number of pieces of information per student might not be a constant number, because, say, we are printing out fields from a related file in which marks on individual examination papers are held.

We shall tackle our simple example in a way that lends itself to more generality later on, and in a form that most database programs should be capable of handling.

We thus prepare a subsidiary file `results.dat` with records of five fields in it. Each student is represented by five lines of this file,

```
Mr
Abraham L
Spriggs
34 Winchester Road\\...\\England
C
```

and the student records appear one after another in this file. Thus both the field and record separators are carriage returns.

TeX itself needs to know three bits of information:

1. the name of the subsidiary file,
2. the fields to read, and
3. the template of the letter.

We pass it this information in the following form

```
\Fields{\Title\Forenames\Surname
   \Address\Grade}
\Merge{results.dat}{%
\begin{letter}{\Title\ \Forenames\
   \Surname\\\Address}
   \opening{Dear \Title\ \Surname,}
   This letter is to inform you
   that you obtained grade \Grade\ in
   your recent examinations.
   \closing{Yours faithfully,}
\end{letter}}
```

LaTeX should open the subsidiary file and, for each set of five parameters, generate a letter in the `dvi` file. When it reaches the end of the merge file, LaTeX should terminate execution of the `\Merge` command and presumably finish the document.

3 A few complications

Looking at the above example in a bit more generality, we see that we are reading records of n fields from the merge file and placing them into a TeX document in such a way that they replace n preassigned control sequences. However, it may happen that the merge file is prepared by humans, who might possibly have inserted some extra blank lines into the file. Again, it could be that certain sorts of fields might be blank, whereas others can never be blank. Perhaps it would be better to build in some degree of error recovery.

We shall make the assumption that the first field in any record is definitely a non-blank one and that we know beforehand whether each of the others might conceivably be blank. We make a modification to our `\Fields` statement. It can contain not only the field name control sequences but also the tokens `+` and `-`, with the following interpretation. A `+` indicates that all following fields should be re-read until a non-blank result is obtained. A `-` indicates that any following fields could conceivably be blank, subject to the restriction that the very first field is always non-blank.

Thus the command

```
\Fields{\a+\b\c-\d}
```

would indicate that only `\d` is allowed to be blank, because the `+` token has no effect. In

```
\Fields{-\a\b+-\c+\d}
```

the initial `-` token enables blank reading of data tokens, but the very first data token is not permitted to be blank anyway. Thus `\a` is read as a non-blank token and `\b` as a possibly blank token. The sequence `+-` now switches non-blank reading on and off again, so `\c` is read as possibly blank. Finally `\d` is non-blank.

Another complication we allow is that the `\Fields` command can appear several times in our file. The interpretation is that the last occurrence of `\Fields` before we encounter the `\Merge` command will indicate the fields to be read for every record. Any occurrences of `\Fields` within the merged text indicate a new list of fields to be read when that command is encountered. This lets us do some conditional processing, such as[1]

```
\ifx\Title\Mrs
   \Fields{\MaidenName}
\fi
```

and also gives us some flexibility about the field order later on.

[1] It is assumed that `\Mrs` expands to `Mrs`.

It should also be stressed that the undefined control sequences appearing in the template need not correspond exactly to the fields in the subsidiary file. An example might be that the subsidiary file contains the text

```
Spriggs, Mr Abraham L
```

and one field read is `\FullName`. TEX would then have to pre-process this name to generate its several components as used in the template. The command `\PreProcess` could be included at the start of the template.

```
\def\parse#1, #2 #3\endparse{%
   \def\Surname{#1}\def\Title{#2}%
   \def\Forenames{#3}}
\def\PreProcess{\expandafter
   \parse\FullName\endparse}
```

An alternative and simpler looking approach to reading fields from a file `\fil` might be to define each such field as follows.

```
\def\Field#1{\def#1{\read\fil to#1#1}}
\Field\Name \Field\Address \Field\Mark
```

The first time `\Name` is encountered, it reads its own expansion from `\fil` and then expands itself. Henceforth, it has acquired its new expansion. The disadvantage is that `\Name` must appear in the text before any subsidiary field such as `\Surname` can be used.

Finally we should consider the possibility that the second parameter of `\Merge` might be too large to fit into memory. We can clearly handle this problem by allowing the second parameter merely to consist of the text `\input template`, so that the root file handles two subsidiary files, one containing the template and the other containing the fields.

4 Implementation of the simple case

For convenience we define a frequently used combination here.

```
\def\glet{\global\let}
```

The subsidiary merge file is defined next. A macro is then defined that attempts to open it for reading. If that is unsuccessful, the file is closed and an error message is issued.

```
\newread\MergeFile
\def\InputFile#1{%
   \openin\MergeFile=#1
   \ifeof\MergeFile
   \errmessage{Empty merge file}%
   \closein\MergeFile
   \long\def\MakeTemplate##1{%
      \def\Template{}}%
   \else\GetInput\fi}
```

The command `\MakeTemplate` will be used later to generate the body of the form into which fields are inserted. We redefine it if the file is empty so that it produces no text.

Because the conditional `\ifeof` does not return true until after an unsuccessful read operation, a mechanism of looking ahead is used which is similar to that found in Pascal.

```
\def\GetInput{{\endlinechar=-1
   \global\read\MergeFile to\InputBuffer}}
```

We set up a mechanism for deciding whether or not we have exhausted the merge file. It forces `\ifeof` to return true by skipping over blank lines.

```
\def\SeeIfEof{%
   \let\NextLook\relax
   \ifeof\MergeFile
   \else
      \ifx\InputBuffer\empty
         \LookAgain
      \fi
   \fi
   \NextLook}
\def\LookAgain{\GetInput
   \let\NextLook\SeeIfEof}
```

We can now prepare to read actual fields from the merge file. A conditional is used to indicate whether or not the field we are about to read is allowed to be blank. We also set up a mechanism for changing its value.

```
\newif\ifNonBlank \NonBlankfalse
\def\AllowBlank{\global\NonBlankfalse}
\def\DontAllowBlank{\global\NonBlanktrue}
```

Fields are actually read by means of the following command. Its only parameter is the name of the control sequence into which the field is read.

```
\def\ReadIn#1{%
   \ifNonBlank\SeeIfEof\fi
   \ifeof\MergeFile
      \gdef#1{??}\MissingField
   \else
      \glet#1\InputBuffer
      \GetInput
   \fi}
\def\MissingField{%
   \message{Missing field in file}}
```

The `\Fields` command places its parameter into a token register called `\GlobalFields`. This command will be redefined by the `\Merge` command.

```
\newtoks\GlobalFields
\def\Fields#1{\GlobalFields{#1}}
```

When a field token list is read, each individual token within it must be either read as a field or

interpreted as a blank/nonblank switch. The next token is then read by tail recursion. It is assumed that the final token in the list is \EndParseFields. This must be defined to expand to something unlikely to be read as a value of one of the fields, and so we \let it to \ParseFields.

```
\def\ParseFields#1{%
   \ifx#1\EndParseFields
      \let\NextParse\relax
   \else
      \let\NextParse\ParseFields
      \ifx#1+\DontAllowBlank
      \else
         \ifx#1-\AllowBlank
         \else\ReadIn#1
         \fi
      \fi
   \fi\NextParse}
\let\EndParseFields\ParseFields
```

We apply this command to our token register after expanding it.

```
\def\ReadFields#1{\expandafter\ParseFields
   \the#1\EndParseFields
   \AllowBlank}
```

At long last we are ready to define the \Merge command itself. The first parameter is the filename of the subsidiary file and the second is the template or form into which fields are inserted. Since a \Fields command within the \Merge text is meant to act immediately on the token list that follows it, we redefine it to operate in a different way.

```
\long\def\Merge#1#2{\begingroup%
   \InputFile{#1}%
   \def\Fields##1{%
      \ParseFields##1\EndParseFields}%
   \MakeTemplate{#2}\Iterate}
\long\def\MakeTemplate#1{\def\Template{#1}}
```

The grouping keeps any changes to the definition of \MakeTemplate local to this merge. Thus several consecutive merges can be handled within one document. The \endgroup is supplied by the macro \Iterate when the merge file has been exhausted.

\Iterate must read the fields which were declared before it was entered, substitute them into its template and repeat itself using tail recursion if the end of the merge file has not been encountered.

```
\countdef\Iteratecounter=9
\Iteratecounter=0
\def\Iterate{%
   \global\advance\Iteratecounter by1
   \ReadFields\GlobalFields
   \Template
   \SeeIfEof
   \ifeof\MergeFile
      \def\NextIteration{%
         \endgroup\closein\MergeFile}%
   \else
      \let\NextIteration\Iterate
   \fi
   \NextIteration}
```

The point of the use of counter 9 in the above is that it is accessible to the print driver for page selection. Anyone who has started printing 150 letters, all with page number 1, only to run out of paper half way, will appreciate the use of this artifice!

5 A complicated example

We will next look at an example in which the template contains a table of indeterminate length, albeit fixed width. So far our macros work in either plain TEX or in LATEX, but the way in which these two packages handle tables is slightly different. However, the only difference that need concern us is that LATEX uses \\ where plain TEX uses \cr.

The example given here is in LATEX, but our style will work equally well in plain TEX. In our student letter we wish to insert a table of course codes and marks. Since each student did a different number of courses, we need some way of recognizing the end of the course list in the merge file. The default will be to insert a blank line at the end of such a sub-list. Thus, the following text appears before the close of the letter template.

```
Here are your marks on individual papers.
\begin{center}
   \begin{tabular}{|lr|}\hline
      Code&Mark\\\hline
      \MultiRead{2}\\\hline
   \end{tabular}
\end{center}
```

The merge file now has the following structure.

```
Title
...
Grade
Code
Mark
...
Code
Mark
⟨blank⟩
Title
...
```

In other applications some of the fields in the table might possibly be blank. We then let the user

change the ⟨*blank*⟩ line marking the end of a list to some other string of his own choosing.

```
\MarkEnd{***}
```

There might be multiple tables in the same template, with their data intermingled in the merge file with main fields. The generalized `\Fields` command allows us to order the merge file however we want. Thus we could have main fields, then a table, followed by more main fields, and so on.

A final complication is that the fields appearing in a table are essentially anonymous. By this I mean that they are transferred into the table as they are, without any pre-processing possible through appearing in the template as control sequences. If we wish what appears in the table to be different from what appears in the file, a mechanism is needed to tell TEX that a certain column has to be treated in a certain way. The command

```
\Process{n}{\foo}
```

will replace every field ⟨*f*⟩ read into column *n* by `\foo{`⟨*f*⟩`}`. It is even possible to do some numerical calculations by this method.

6 Implementation of merged tables

We set up two counters, one for the column we are reading and the other for the total number of columns in the table. We also need a conditional to mark the start of the table, so that we terminate each row correctly with `\\` or `\cr`, or nothing at all at the beginning of the first row.

```
\newcount\MultiCount \newcount\MaxCount
\newif\ifStartOfList
```

The parameter to `\MultiRead` is the number of columns to read at a time. This command passes control to `\NextRead` after initializing certain parameters.

```
\def\MultiRead#1{%
   \ifnum#1>0
      \SelectCR
      \MakeEmpty{#1}%
      \global\StartOfListtrue
      \glet\NextRead\MRead
      \AllowBlank
      \global\MaxCount=#1
      \NextRead
   \fi}
```

The command `\MakeEmpty` is required by the preprocessing of each field. The idea is that the command `\csname` pr*nn*`\endcsname`, which we will loosely call `\pr`*nn*, is executed on each field in column *nn*. However, most of these commands will be undefined, and so we equate each of those that has not been defined to `\empty`.

```
\newcount\Emptyctr
\def\MakeEmpty#1{\Emptyctr=0
   \loop
      \advance\Emptyctr by1
      \expandafter\ifx\csname
         pr\the\Emptyctr\endcsname\relax
         \expandafter\glet\csname
            pr\the\Emptyctr\endcsname\empty
      \fi
      \ifnum\Emptyctr<#1
   \repeat}
```

Note that, because of the way we are accessing it via `\csname`, the first time `\pr`*nn* is encountered it equates to `\relax`.

The command `\Process#1#2` defines `\pr#1` to mean `#2`.

```
\def\Process#1#2{%
   \expandafter\def\csname
      pr#1\endcsname##1{#2{##1}}}
```

We need to know how the last row is to be recognized. The default is an empty line in the merge file.

```
\def\MarkEnd#1{\gdef\EndMarker{#1}}
\MarkEnd{}
```

We collect each row in a token register. The full row is assembled in `\NextLine` before being passed back to TEX. Each field is read in `\TempField` and then placed temporarily into `\NextField`.

```
\newtoks\NextLine \newtoks\NextField
```

It is not necessary to do things this way; `\edef` can be used instead, but that approach might expand tokens prematurely.

After the next field has been read, it is appended to `\NextLine`.

```
\def\AppendNextField{%
   \global\advance\MultiCount1
   \NextField=\expandafter{\TempField}%
   \edef\Append{\NextLine=
      {\the\NextLine&\csname
         pr\the\MultiCount\endcsname
         {\the\NextField}}}%
   \Append}
```

We need to insert the correct end marker after each row of the table. The token `\cr` must be disguised a little before it is acceptable in a LATEX document.

```
\def\SelectCR{%
   \ifx\array\undefined
      \gdef\EndLine{\cr}%
```

```
   \else
      \glet\EndLine\\%
   \fi}
\def\FinishLine{%
   \ifStartOfList
      \global\StartOfListfalse
   \else\EndLine\fi}
```

This makes the assumption that if \array is defined then we must be in LaTeX.

We need a command to finish off a table. This should reset \NextRead to \AllowBlank to terminate the tail recursion, and also do some error recovery in case the file ends prematurely in the middle of a row.

```
\def\StopProcessing{%
   \global\MultiCount\MaxCount
   \glet\NextRead\AllowBlank}
```

The command \MRead prepares to read a row of a table. It reads a field from the merge file and checks to see whether the table has been exhausted.

```
\def\MRead{%
   \global\MultiCount=1
   \ReadIn\TempField
   \ifx\TempField\EndMarker
      \StopProcessing
   \else
      \FinishLine
      \NextField=\expandafter{\TempField}%
      \edef\StartLine{\NextLine={\csname
         pr1\endcsname{\the\NextField}}}%
      \StartLine
      \ConstructNextRow
   \fi
   \NextRead}
```

Command \ConstructNextRow does most of the work of assembling a row of the table. It assembles \MaxCount fields at a time into \NextLine unless an error is encountered.

```
\def\ConstructNextRow{%
   \loop
      \ReadIn\TempField
      \ifx\TempField\EndMarker
         \glet\TempField\empty
         \StopProcessing
         \MissingField
      \else
         \ifeof\MergeFile
            \glet\TempField\empty
            \StopProcessing
            \MissingField
         \fi
      \fi
      \AppendNextField
      \ifnum\MultiCount<\MaxCount
   \repeat
   \the\NextLine}
```

7 A final example

Here is a LaTeX example to illustrate the table processing features of textmerg.sty.

```
\documentstyle[12pt,textmerg]{article}
\MarkEnd{***}
\Process{2}{\Advance}
\def\Advance#1{#1\addtocounter{page}{#1}}
\Fields{+\Name\Verb}
\begin{document}
\Merge{silly.dat}{%
   Dear \Name,\par
   Here is a table to \Verb\ at:
   \Fields{\Width}%
   \begin{tabular}{*{\Width}c}
      \MultiRead\Width
   \end{tabular}.\par
   \Fields{\Adj}%
   That was \Adj!\clearpage}
\end{document}
```

The effect of this file is not apparent until we see silly.dat. It is listed here in four columns.

Mike	3	good	22
look	11	Shelagh	23
3	12	gaze	24
1	13	2	***
2	***	21	horrid

References

[Bel87] Edwin V. Bell, II. AutoLetter: A TeX form letter procedure. *TUGBoat*, 8(1):54, April 1987.

[Gar87] John S. Garavelli. Form letter macros. *TUGBoat*, 8(1):53, April 1987.

[Lee86] John Lee. Form letters. *TUGBoat*, 7(3):187, October 1986.

[McK87] Graeme McKinstry. Form letters. *TUGBoat*, 8(1):60, April 1987.

◇ Mike Piff
Department of Pure Mathematics
University of Sheffield
Sheffield S10 2TN
England
Janet: M.Piff@shef.ac.uk

A style file for printing sheets of labels

Sebastian Rahtz

Contents

Abstract

A LaTeX style to print a regular grid of labels on a page, suitable for sheets of labels which can be fed through a laser printer. Macros are provided to allow easy input of names and addresses in a form free of TeX markup.

1 Usage

This style file was written to print labels from the shop around the corner from me. These have 8 rows and 3 columns on a sheet of A4 paper. Your labels will very likely be different. So first you have to tailor this file to your particular type of label. Edit the lines below which look like this:

```
\num@labelcols=3
\num@labelrows=8
```

to reflect *your* grid (maybe you have only two columns of ten labels each, for instance). Now make sure that your printer driver prints the page *exactly* as it should in vanilla TeX, i.e. with the origin of the page down 1in and right 1in from the top left hand corner of the paper. If it doesn't, adjust your driver parameters, or edit the settings below where I take 1in off the margins. The most likely problem with these macros is that you will have contents which are quite wide, and which therefore need to use the very edges of the paper, on which your printer may not write correctly. Little one can do about this — use a small point size.

The simplest form of input is very easy, as in the following example:

```
\documentstyle{labels}
\begin{document}
\begin{labels}
\input names.dat
\end{labels}
\end{document}
```

where `names.dat` contains names and address in plain format with simply a blank line between entries. You can, of course, just have the names and addresses in the main file, rather than using `\input` to include them. If the file ends in blank lines, expect problems — sorry! Use your editor...

But there are also other ways off accessing the same system:

1. by having entries like this:

   ```
   \addresslabel{me\\
   here and there\\
   england\\
   }
   ```

   without the `labels` environment.

2. if you have labels in the simple format in a file, just write a `.tex` file like this:

   ```
   \documentstyle{labels}
   \begin{document}
   \labelfile{filename}
   \end{document}
   ```

   and all will be done for you.

3. if you want to *duplicate* the label, there is a counter called `\numberoflabels` which you can set, so

   ```
   \numberoflabels=4
   \addresslabel{Me \\my street
           \\ mytown \\ England}
   ```

   will print the address 4 times in a row

4. For more sophisticated users, there is a macro `\genericlabel` which you can call, with an argument of whatever you want to appear on the label (e.g. for disk labels, etc.). Thus you could have

   ```
   \genericlabel{%
   \begin{tabular}{c}
   \hline
   My Amazing Program\\
   \hline
   Disk 1 of 1
   \hline
   \em We aim to serve\\
   \end{tabular}
   }
   ```

   to produce a label like this:

   ```
   |------------------|
   |My Amazing Program|
   |------------------|
   |Disk 1 of 1       |
   |------------------|
   |We aim to serve   |
   ```

In all modes, you can opt for a frame around each label by setting a Boolean variable called 'framedlabels', e.g.

```
\framedlabelstrue
```

By default you get no frames — I am not sure when you *would* want frames, but who knows.

2 The utility macros

First of all, identify what is happening.

```
\def\fileversion{v4}
\def\filedate{92/1/1}
\immediate\write\sixt@@n{File: 'labels.sty'
\fileversion\space <\filedate> (SPQR)}
```

Now take a copy all of 'article' style to start with, just in case any of it is needed (probably not, but you never know).

```
\input article.sty
```

We will be recording the size of a label, and the dimensions of the grid, so set up variables accordingly.

```
\newdimen\label@width
\newdimen\label@height
\newcount\num@labelcols
\newcount\num@labelrows
\newdimen\left@border
\newdimen\top@border
\newdimen\half@label
\newdimen\area@width
\newsavebox{\this@label}
\newcount\label@number
\newcount\numberoflabels
\newcount\l@so@far
\newif\iffframedlabels
\newif\iffirst@label
\first@labeltrue
\framedlabelsfalse
```

The user will probably need to change the following values to reflect the style of labels in use.

```
\num@labelcols=3
\num@labelrows=8
```

Variables are provided to allow you to force a border on the left edge of labels, in case you do not want to print right to the edge, and at the top; these values will affect every label, of course, so you may need to experiment to get pleasing results. 8mm is the amount my LaserJetIII seems to ignore on the left.

```
\left@border=8mm
\top@border=4mm
```

We need to reset all the dimensions appropriately for an A4 page of labels, and the printer will need to know about A4 as well. Obviously if you use a different page size, you will need to alter things here. Some of these changes may be printer dependent. This should all mean we are actually dealing with the whole bit of paper.

```
\textwidth=210mm
\textheight=297mm
\topmargin=-1in
\headheight=0em
```

```
\headsep=0em
\topskip=0em
\footskip=0em
\footheight=0em
\oddsidemargin=-1in
\evensidemargin=-1in
\pagestyle{empty}
\parindent=0em
\parskip=0pt
```

Now calculate the size of labels simply as a proportion of the page size (if you haven't got that right, this won't work, will it?).

```
\label@width\textwidth\divide\label@width by\num@labelcols
\label@height\textheight\divide\label@height by\num@labelrows
\typeout{Creating labels sized \the\label@width\space by \the\label@height}
\label@number=1
```

It is not usually advisable to make the label printing go right to the edge of the available area, so 'area@width' gives the area that will actually be used for printing; the width is cut down by whatever we gave as 'left@border'. It can always be set to 0 if you have a design that uses the whole label.

```
\area@width=\label@width%
\advance\area@width by -\left@border%
\half@label=\label@height\divide\half@label by 2
\advance\half@label by -\top@border
```

We might want to print the same label several times, so `\sticky@label` will repeat `\make@label` a specified number of times (`\numberoflabels`)

```
\numberoflabels=1%
```

```
\def\sticky@label{\l@so@far=0%
\loop\ifnum\l@so@far<\numberoflabels\advance\l@so@far by 1\make@label%
\repeat}
```

The real label-making macro, which assumes the actual text is in a box called `\this@label`. It is vital to make sure spaces are not included at the end of lines in these macros, or all hell breaks loose.

```
\def\make@label{%
\iframedlabels%
\let\boxing@type\framebox%
\else%
\let\boxing@type\makebox%
\fi%
\boxing@type[\label@width][c]{%
\rule{0pt}{\label@height}%
```

We set a position to half-way up a strut of the height of the label, so forcing text to be the right height and vertically centred.

```
\raisebox{\half@label}[0pt][0pt]{%
\rule{\left@border}{0pt}\usebox{\this@label}}}%
```

We only start a new line if we have printed a row of `\num@labelcols` labels

```
\ifnum\label@number=\num@labelcols%
\endgraf\nointerlineskip%
\label@number=1\else\advance\label@number by 1\fi%
}%
```

Now some macros to allow 'verbatim' names and addresses separated by blank lines. First we need some hackery from Phil Taylor to redefine end of line; define carriage-return to check what the next token is; if its another `^M` then we have a blank line.

```
\catcode `\^^M = \active
```

```
\def ^^M{\futurelet\nexttoken\isitapar}%
\def\isitapar{\ifx^^M\nexttoken\let\action=\new@label%
\else\let\action\start@newline\fi\action}%
```

If we have met a blank line, finish current label and start a new one. swallow pending ^M, or we will have a blank line at the start of each label

```
\def\new@label{\message{+}\end@@label\start@@label\@gobble}%
```

Otherwise just start a new line

```
\def\start@newline{\expandafter\newline}%
\def\startingtoken{\ifx^^M\firsttoken\let\action=\@gobble\else%
\let\action=\relax\fi\action}%
```

Re-instate the original catcode for carriage-return

```
\catcode `\^^M = 5\relax%
```

Define macros to call at beginning and end of labels, to set things up properly.

```
\def\start@@label{%
\savebox{\this@label}\bgroup\raggedright%
\begin{minipage}{\area@width}%
\catcode `\^^M =\active}%
\def\end@@label{%
\end{minipage}\egroup\sticky@label}%
```

3 User macros

The basic case is a generic macro which takes its argument and puts it out on a label.

```
\def\genericlabel#1{%
\iffirst@label\iframedlabels%
\advance\label@height by-2\fboxsep%
\advance\label@height by-2\fboxrule%
\half@label\label@height\divide\half@label by 2
\advance\half@label by -\top@border%
\first@labelfalse%
\fi\fi%
\savebox{\this@label}{#1}\sticky@label}
```

For compatibility with an old label style, lines ending in // and marked with \addresslabel{....}

```
\def\addresslabel#1{\genericlabel{%
\begin{tabular}{l}#1\end{tabular}}}
```

Now easier environments for verbatim labels. If we want framed labels, we need to adjust the width available to use to allow for the rule width and the gap between box and rule, in both axes. This is doubled up, as it happens on both sides / bottoms. We have to check in case the first \begin{labels} has a ^^M after it or (preferably) is terminated by a %

```
\newenvironment{labels}%
{%
\iffirst@label\iframedlabels%
\advance\area@width by-2\fboxsep%
\advance\area@width by-2\fboxrule%
\advance\label@height by-2\fboxsep%
\advance\label@height by-2\fboxrule%
\half@label\label@height\divide\half@label by 2
\advance\half@label by -\top@border%
\first@labelfalse%
\fi\fi%
\start@@label\futurelet\firsttoken\startingtoken}%
{\end@@label}
```

Even more foolproof: simply take a parameter of file name

```
\def\labelfile#1{\begin{labels}\input#1\end{labels}}
```

or prompt for it:

```
\def\promptlabels{\typein[\labelfilename]{What is the name of the
label file?}
\labelfile{\labelfilename}}
```

4 History and acknowledgements

- v.1 May 9th 1989 simply allowed for `\addresslabel{... \\ ...\\...}`
- v.2 July 15th permitted verbatim style with no explicit end of lines
- v.3 March 1991 made more generic
- v.4 January 1992 checked and made to work with emtex drivers to my satisfaction, and documented to bare-bones level with 'doc' system.

The crucial macros which make the system bearable for mailing lists by redefining end of line came from Phil Taylor; apologies to him for using them in a LaTeX style file!

◇ Sebastian Rahtz
ArchaeoInformatica
12 Cygnet Street
York Y02 1AG
spqr@uk.ac.york.minster

Abstracts

Les Cahiers GUTenberg
Contents of Recent Issues

Numéro 13 – juin 1992

Bernard GAULLE, Éditorial : morosité francophone ou récession économique ? [French moodiness or economic recession?]; pp. 1–4

GUTenberg's president provides ample evidence of a strong and dynamic (LA)TEX community, in spite of the low interest shown for the 1991 GUTenberg meeting and other reasons which caused the board to decide to skip GUTenberg'92. While the two meetings seemed to have all the best possible organisation and components, participation didn't seem to match the efforts made. And yet the intensity of work being done on various fronts (e.g., LaTeX3), the increasing number of articles on (LA)TEX in various national magazines and journals, the increase in requests to GUTenberg for information, courses and subscriptions to the *Cahiers*, all show that things are humming. It would therefore seem that the economy, not a downturn in interest (along with perhaps too many TEX meetings), is the most likely cause for reduced participation at recent meetings.

Hans Ed. MEIER, Régles fondamentales de mise en page [Basic rules for page layout]; pp. 5–38

Originally published in 1991 in German, this article as translated has been augmented with comments on French typographic style. Meier, described in the editorial as a retired typographer, discusses in his article design elements not only of

books but all manner of documents, from in-house reports to business cards. The article comprises two parts: a section (pp. 6–14) on various elements of a document (e.g., spacing, margins, justification, use of small caps, centering, the general look of a page, etc.), followed by before/after examples taken from documents Meier had worked on while at the Swiss Federal Institute of Technology in Zurich (pp. 14–38). The first section provides very useful basic notions (as promised in the title) on style and layout, and with the additional comments from the GUTenberg editorial staff, this portion is very interesting and informative for those interested in variations across linguistic and cultural groups. The second section, while its rather dogmatic rejection and improvements may not sit well with everyone, is nevertheless a good baseline which users new to the notions of page layout and document design could either choose to adhere to, or deviate from. The article has a short bibliography of 6 titles in German, with 2 references in French, added by the GUTenberg editorial staff.

Emmanuel SAINT-JAMES, Une police pour la science. De l'impact du traitement de texte sur l'activité scientifique [Guidelines for science: the impact of word processing on scientific activity]; pp. 39–54

The editorial for this issue introduces the Saint-James article as one which is more philosophical — even epistemological — and as with the previous article, one not specifically aimed at TEX at all. "Difficult to read, sometimes obscure and often provocative, each paragraph prompts one to stop and think" is how the editorial describes the paper, and the argumentation is indeed difficult to follow at times. The paper is a result of the author's efforts in producing a 400-page publication (*Traité de programmation*), and discusses the choices at various levels: characters (font selection), lexicon (at issue, the infusion/intrusion of English computer terminology), the document structure, and a mini-diatribe about bibliographies citing "old news", for which he has little use. Unfortunately, this disdain is also apparent in the references which accompany the article (in fact, the author explicitly refers to this "concession"!), so the reader is unable to readily pursue the casual asides which pass for references. But this should not deter those who can read French from approaching the article and reading an account which resides more at the meta level than at the mere production level.

Bobby BODENHEIMER, TEX, LATEX, etc : questions et réponses [French adaptation and translation of an issue of the "FAQ" — Frequently Asked Questions — posted 2 March 1992 to `comp.text.tex`, v.1.22]; pp. 55–77

An excellent idea [one which TTN might consider!] of presenting some of the main items in a recent issue of the Frequently Asked Questions file which is posted monthly to the `c.t.t` newsgroup, the GUTenberg editors have produced a handy reference article for the many users of (LA)TEX not able to access the Internet. The main areas treated include: archives (where to find what); drivers (queries for specific printers); graphics programs (where to get them, how to use them); fonts (working with METAFONT, using PS fonts, converting font formats, locating different fonts); "How do you do x in TEX?" (music, an index); conversions (e.g., x to LATEX, LATEX to x); hyphenation problems; error messages; "How do you do y in LATEX?" (style files, various specific questions); and finally, user groups and documentation. The FAQ closes with a pertinent reminder that electronic accessing via ftp is a privilege, not to be abused. A welcome addition is the Index to the FAQ which the editors of GUTenberg have provided, making this a doubly useful article.

Annonces et publicités [Announcements and advertisements]; pp. 78–90

The final portion of the issue carries a number of items, including a listing of TEX versions available from GUTenberg for various systems (Mac, PCs, VAX/VMS, and UNIX variations); the detailed announcement of the Prague EuroTEX'92 meeting (including registration form); and the announcement for TUG'92 in Portland. As well, there are advertisements from Y&Y (fonts), Cambridge University Press (their Cambridge Electronic Publishing service), and ET, a TEX text editor for the PC. A loose flyer contains advertisements for *Textures* on one side, and MacFEM (a program "for solving partial derivative equations") on the other.

Calendar

1993

Jan 7 TEX–Stammtisch at the Universität Bremen, Germany. For information, contact Martin Schröder (`l15d@alf.zfn.uni-bremen.de`; telephone 0421/6289813).

Jan 21 MITTUG meeting, Massachusetts Institute of Technology, Cambridge, Massachusetts. Room 1-134, 12:00-13:00. For information, contact Robert Becker (robertb@math.mit.edu; 617-253-1797).

Jan 21 TEX-Stammtisch in Duisburg. For information, contact Friedhelm Sowa (`tex@ze8.rz.uni-duesseldorf.de`, telephone 0211/311 3913).

Jan 27 TEX–Stammtisch, Hamburg, Germany. For information, contact Reinhard Zierke (`zierke@informatik.uni-hamburg.de`; telephone (040) 54715-295).

Feb 1–2 Symposium: The Ethics of Scholarly Publishing, Toronto Hilton Hotel, Toronto, Ontario. For information, contact L. Forget, Conference Services Office, National Research Council, Ottawa, Ontario K1A 0R6 (613-993-9009; Fax: 613-957-9828).

Feb 4 TEX–Stammtisch at the Universität Bremen, Germany. (For contact information, see Jan 7.)

Feb 16 ***TUGboat* Volume 14, 2nd regular issue:** Deadline for receipt of *technical* manuscripts (tentative).

Feb 18 MITTUG meeting, Massachusetts Institute of Technology, Cambridge, Massachusetts. Room 37-252, 12:00-13:00. (For contact information, see Jan 21.)

Feb 18 TEX-Stammtisch in Duisburg. (For contact information, see Jan 21.)

Feb 22 **Papers for TUG Annual Meeting**, deadline for submission of proposals (see *TUGboat* 13, no. 3, p. 398).

Feb 24 TEX–Stammtisch, Hamburg, Germany. (For contact information, see Jan 27.)

Feb 24–27 CONCEPPTS 93, The Prepublishing Conference, Orange County Convention Center, Orlando, Florida. "International Conference on Computers and Electronic Publishing and Printing Technologies". For information, phone: 703-264-7200, Fax: 703-620-9187.

Mar 1 **Donald E. Knuth Scholarship**, deadline for submission of projects. (See *TUGboat* 13, no. 3, pp. 395–396.)

Mar 4 TEX–Stammtisch at the Universität Bremen, Germany. (For contact information, see Jan 7.)

TUG Courses, San Francisco, California

Mar 1–5 Intensive LATEX

Mar 8 TEX for Publishers

Mar 9–10 Practical SGML and TEX

Mar 9 ***TUGboat* Volume 14, 1st regular issue:** Mailing date (tentative).

Mar 9–12 DANTE'93 and General Meeting, Chemnitz, Germany. For information, contact Dr. Wolfgang Riedel (`wolfgang.riedel@hrz.tu-chemnitz.de`).

Mar 16 ***TUGboat* Volume 14, 2nd regular issue:** Deadline for receipt of news items, reports (tentative).

Mar 18 MITTUG meeting, Massachusetts Institute of Technology, Cambridge, Massachusetts. Room 37-252, 12:00-13:00. (For contact information, see Jan 21.)

Mar 18 TEX-Stammtisch in Duisburg. (For contact information, see Jan 21.)

Mar 22 **TUG Course:** TEX for Publishers, Boston, Massachusetts.

Status as of 10 November 1992

Mar 30–31 UK TEX Users' Group, Glasgow, Scotland. (Note: this is just before the BCS EPSG meeting.) Topics: METAFONT, theoretical and practical; and font selection schemes, virtual fonts, multiple languages and hyphenation, etc. — everything you need to know to use TEX to typeset non-American languages. For information, contact Phil Taylor (chaa006@vax.rhbnc.ac.uk).

Mar 31 TEX–Stammtisch, Hamburg, Germany. (For contact information, see Jan 27.)

Apr 1 TEX–Stammtisch at the Universität Bremen, Germany. (For contact information, see Jan 7.)

Apr 15 TEX-Stammtisch in Duisburg. (For contact information, see Jan 21.)

TUG Courses, Boston, Massachusetts

Apr 19–23 Beginning/Intermediate TEX

Apr 26–30 Intensive LATEX

Apr 28 TEX–Stammtisch, Hamburg, Germany. (For contact information, see Jan 27.)

May UK TEX Users' Group, Chichester, England. Visit to John Wiley & Sons Ltd. Host: Geeti Granger. For information, contact David Murphy (D.V.Murphy@computer-science.birmingham.ac.uk).

May 6 TEX–Stammtisch at the Universität Bremen, Germany. (For contact information, see Jan 7.)

May 20 TEX-Stammtisch in Duisburg. (For contact information, see Jan 21.)

May 25 ***TUGboat* Volume 14, 2nd regular issue:** Mailing date (tentative).

May 26 TEX–Stammtisch, Hamburg, Germany. (For contact information, see Jan 27.)

Jun 3 TEX–Stammtisch at the Universität Bremen, Germany. (For contact information, see Jan 7.)

Jun 6–10 Society for Technical Communication, 40th Annual Conference. Dallas, Texas. For information, contact the Society headquarters, 901 N. Stuart St., Suite 904, Arlington, VA 22203-1854. (703-522-4114; Fax: 703-522-2075), or the Conference Manager, Binion Amerson (aba@oc.com).

Jun 9 **TUG Course:** TEX for Publishers, New York City.

Jun 10 11th NTG Meeting, "From Font to Book", Royal Dutch Meteorological Institute, De Bilt, Netherlands. For information, contact Theo Jurriens (taj@astro.rug.nl).

TUG Courses, San Diego, California

Jun 7–11 Modifying LATEX Style Files

Jun 14–18 Beginning/Intermediate TEX

Jun 21–25 Advanced TEX and Macro Writing

Jun 17 TEX-Stammtisch in Duisburg. (For contact information, see Jan 21.)

Jun 30 TEX–Stammtisch, Hamburg, Germany. (For contact information, see Jan 27.)

Jul 26–29 **TUG Annual Meeting: "World Wide Window on TEX"**, Aston University, Birmingham, U.K. For information, contact the organizing committee: Chris Rowley (C.A.Rowley@open.ac.uk) or Malcolm Clark (malcolmc@wmin.ac.uk), or the TUG office.

Aug 9–13 **TUG Course:** Beginning/Intermediate TEX, Boston, Massachusetts

Aug 17 ***TUGboat* Volume 14, 3rd regular issue:** Deadline for receipt of *technical* manuscripts (tentative).

Aug 23–27 **TUG Course:** Intensive LATEX, Ottawa, Canada

Sep 14 ***TUGboat* Volume 14, 3rd regular issue:** Deadline for receipt of news items, reports (tentative).

Sep	23–24	**TUG Course:** Book and Document Design with TEX, Boston, Massachusetts
Oct	18–22	**TUG Course:** Beginning/Intermediate TEX, Chicago, Illinois

TUG Courses, Boston, Massachusetts

Oct	25–29	Intensive LATEX
Nov	1–5	Advanced TEX and Macro Writing
Nov	8–9	Practical SGML and TEX

Nov	10	**TUG Course:** SGML and TEX for Publishers, New York City
Nov	12	**TUG Course:** TEX for Publishers, Washington, DC
Nov	18	12th NTG Meeting, "(LA)TEX user environment", Oce, Den Bosch, Netherlands. For information, contact Gerard van Nes (vannes@ecn.nl).
Nov	23	***TUGboat* Volume 14, 3rd regular issue:** Mailing date (tentative).

For additional information on the events listed above, contact the TUG office (805-899-4673, email: tug@math.ams.com) unless otherwise noted.

Call for Papers: Special Issue of *Electronic Publishing: Origination, Dissemination and Design* on Active Documents

Electronic Publishing: Origination, Dissemination and Design (*EP–odd*) is pleased to announce a special issue on the topic of Active Documents, planned to appear in 1993.

The presentation and contents of an active document are dependent on computation. An active document can be intended for interactive presentation or for paper presentation. A paper-based active document's content might be generated by execution of an algorithm when the document is assembled. The organization of an interactive active document might depend on characteristics of the computer environment in which it is displayed. In either case, the key characteristics are that the document has been designed by an author with a particular purpose in mind and that the document's content and/or structure responds to aspects of the surrounding world's state.

We are interested in papers on all aspects of active documents. As a very general starting point, some possible topics include system design, underlying support issues, evaluation, language issues (especially the representation of the activity within the system), and cognitive issues.

As a rough guideline, papers should be in the range of 10 to 20 pages in length, although longer and shorter papers are appropriate if dictated by the subject matter. All papers will be refereed using the normal *EP–odd* refereeing process. Manuscripts must be submitted in final form as the limited time available for reviewing the contributions to the special issue will not permit very many passes through the editing cycle. Authors will be asked to follow the *EP-odd* author guidelines, (four copies of the manuscript are requested), and they are asked to include a copy of the *EP–odd* copyright release form with their submission. Copies of the guidelines and copyright release form may be found in each issue of *EP–odd* or may be obtained by mail from the guest editor (address below).

Because space is limited in the special issue, it may not be possible to publish all deserving papers. Unless authors request otherwise, such deserving papers will be referred to the normal *EP–odd* editorial process and will be considered for publication in a subsequent issue.

This issue will be coordinated by Vincent Quint, member of the editorial board of *EP–odd*. He may be contacted at:

Vincent Quint
INRIA/IMAG
2, rue de Vignate
F-38610 Gieres
France
e-mail: quint@imag.fr
Telephone: +33 76 63 48 31
Fax: +33 76 54 76 15

In order to receive maximum consideration, papers should be received by

April 1, 1993

Papers received after this date will be accepted, but because of schedule constraints may not be able to appear in the special issue.

Late-Breaking News

Production Notes

Barbara Beeton

Input and input processing

Electronic input for articles in this issue was received by e-mail, on diskette, and was also retrieved from remote sites by anonymous ftp. In addition to text, the input to this issue includes METAFONT source code and several encapsulated PostScript files. More than 75 files were used directly to generate the final copy; over 100 more contain earlier versions of articles, auxiliary information, and records of correspondence with authors and referees. These numbers represent input files only; `.dvi` files, device-specific translations, and fonts (`.tfm` files and rasters) are excluded from the total.

Most articles as received were fully tagged for *TUGboat*, using either the `plain`-based or LaTeX conventions described in the Authors' Guide (see *TUGboat* 10, no. 3, pages 378–385). Several authors requested copies of the macros (which we were happy to provide); however, the macros have also been installed at `labrea.stanford.edu` and other good archives, and an author retrieving them from an archive will most likely get faster service. Of course, the TUG office will provide copies of the macros on diskette to authors who have no electronic access.

Both the number of articles and the number of pages in this issue are just over half in LaTeX. In organizing the issue, attention was given to grouping bunches of `plain` or LaTeX articles, to yield the smallest number of separate typesetter runs, and the least amount of handwork pasting together partial pages. This also affected the articles written or tagged by the staff, as the conventions of `tugboat.sty` or `ltugboat.sty` would be chosen depending on what conventions were used in the preceding and following articles; no article was changed from one to the other, however, regardless of convenience.

Font work was required for several articles: Žubrinic on the Glagolitic alphabet (p. 470), Sauter on postal barcodes (p. 472), and two by Haralambous — the `mactt` font (p. 476) and Greek hyphenation (p. 457); this last article also required the new font selection scheme (NFSS).

The article by Kelly and Bischof (p. 443) incorporates several (encapsulated) PostScript images; although it was the only article expressly requiring PostScript processing, several others were also output to PostScript devices for convenience.

The following articles were prepared using the `plain`-based `tugboat.sty`:

- all articles in General Delivery.
- Richard Palais, *Moving a fixed point*, page 425.
- Nigel Chapman, *Searching in a DVI file*, page 447.
- the *Hyphenation exception log*, page 452.
- Darko Žubrinic, *The exotic Croatian Glagolitic alphabet*, page 470.
- John Sauter, *Postnet codes using* METAFONT, page 472.
- two book reviews:
  Arvind Borde, *TEX by Example*, page 487,
  André Heck, ed., *Desktop Publishing in Astronomy & Space Sciences*, page 489.
- Erich Neuwirth, *TEX implementations for IBM PCs: comparative timings*, page 490.
- Paul Anagnostopoulos, *ZzTEX: A macro package for books*, page 497.
- Jonathan Fine, *The* `\noname` *macros — A technical report*, page 505.
- abstracts of the *Cahiers GUTenberg*, page 528.
- the TUG calendar, page 530.
- announcement of the *EP-odd* special issue, page 532.
- these Production notes
- "Coming next issue"

Output

The bulk of this issue was prepared at the American Mathematical Society from files installed on a VAX 6320 (VMS) and TEX'ed on a server running under Unix on a Solbourne workstation. Most output was typeset on an APS-μ5 at the AMS using resident CM fonts and additional downloadable fonts for special purposes. Three articles were output on the Math Society's Compugraphic 9600 Imagesetter: Taylor (p. 433), Kelly and Bischof (p. 443, which contained encapsulated PostScript images), and Haralambous on Greek hyphenation (p. 457).

One photograph, photographically screened in the traditional manner, accompanies the interview with Donald Knuth (p. 419).

The output devices used to prepare the advertisements were not usually identified; anyone interested in determining how a particular ad was prepared should inquire of the advertiser.

Coming Next Issue

Anchored Figures at Either Margin

A figure in a box can be placed in text at one margin or the other, by measuring the box and adjusting the paragraph shape parameters so as to allow room for it. Macros that try to accomplish this automatically must be resourceful enough to decide what to do in a variety of special circumstances; the correctness or appropriateness of each decision depends on the requirements of the user. Daňiel Comenetz presents his solution to the problems that arise in mathematics texts. [Delayed by technical difficulties.]

FIFO and LIFO sing the BLUes

Kees van der Laan presents an exposition on FIFO, First-In-First-Out, and LIFO, Last-In-First-Out, well-known techniques for handling sequences. In TEX macro writing these techniques are used abundantly but are often not easily recognized as such. TEX templates for FIFO and LIFO are given and their use illustrated. Their relation to several techniques presented by Knuth in *The TEXbook* is described.

A Multimedia Document System Based on TEX and DVI Documents

R. A. Vesilo and A. Dunn examine the development of a multimedia document system based on TEX. Multimedia document systems involve many complex components including editors, formatters, display systems and components to support the different media. By using TEX to do the formatting, using a standard text editor to enter the document text contents and define the document structure, and modifying a DVI previewer to include support for non-text contents, the amount of effort required to develop a multimedia document system is greatly reduced. [Delayed by technical difficulties.]

Using TEX to make Agendas and Calendars with Astronomical Events

Inspired by the agenda distributed at the Cork TEX meeting, Jordi Saludes has created macros that will produce calendars and agendas in several formats, allowing the user to include both 'fixed' and 'movable' events. Events of the latter type depend on astronomical phenomena related to the sun and moon, and include religious feast days and phases of the moon; all the necessary calculations are done by TEX based on traditional algorithms.

Institutional Members

The Aerospace Corporation,
El Segundo, California

Air Force Institute of Technology,
Wright-Patterson AFB, Ohio

American Mathematical Society,
Providence, Rhode Island

ArborText, Inc.,
Ann Arbor, Michigan

ASCII Corporation,
Tokyo, Japan

Beckman Instruments,
Diagnostic Systems Group,
Brea, California

Belgrade University,
Faculty of Mathematics,
Belgrade, Yugoslavia

Brookhaven National Laboratory,
Upton, New York

Brown University,
Providence, Rhode Island

California Institute of Technology,
Pasadena, California

Calvin College,
Grand Rapids, Michigan

Carleton University,
Ottawa, Ontario, Canada

Centre Inter-Régional de
Calcul Électronique, CNRS,
Orsay, France

CERN, *Geneva, Switzerland*

College Militare Royal de Saint
Jean, *St. Jean, Quebec, Canada*

College of William & Mary,
Department of Computer Science,
Williamsburg, Virginia

Communications
Security Establishment,
Department of National Defence,
Ottawa, Ontario, Canada

Construcciones Aeronauticas, S.A.,
CAE-Division de Proyectos,
Madrid, Spain

Cornell University,
Mathematics Department,
Ithaca, New York

DECUS, Electronic Publishing
Special Interest Group,
Marlboro, Massachusetts

Department of National Defence,
Ottawa, Ontario, Canada

Digital Equipment Corporation,
Nashua, New Hampshire

E. S. Ingenieres Industriales,
Sevilla, Spain

Edinboro University
of Pennsylvania,
Edinboro, Pennsylvania

Elsevier Science Publishers B.V.,
Amsterdam, The Netherlands

European Southern Observatory,
Garching bei München,
Federal Republic of Germany

Fermi National Accelerator
Laboratory, *Batavia, Illinois*

Florida State University,
Supercomputer Computations
Research, *Tallahassee, Florida*

Fordham University,
Bronx, New York

General Motors
Research Laboratories,
Warren, Michigan

GKSS, Forschungszentrum
Geesthacht GmbH,
Geesthacht, Federal Republic of
Germany

Grinnell College,
Computer Services,
Grinnell, Iowa

Grumman Aerospace,
Melbourne Systems Division,
Melbourne, Florida

GTE Laboratories,
Waltham, Massachusetts

Hughes Aircraft Company,
Space Communications Division,
Los Angeles, California

Hungarian Academy of Sciences,
Computer and Automation
Institute, *Budapest, Hungary*

IBM Corporation,
Scientific Center,
Palo Alto, California

Institute for Advanced Study,
Princeton, New Jersey

Institute for Defense Analyses,
Communications Research
Division, *Princeton, New Jersey*

Intevep S. A., *Caracas, Venezuela*

Iowa State University,
Ames, Iowa

The Library of Congress,
Washington D.C.

Los Alamos National Laboratory,
University of California,
Los Alamos, New Mexico

Louisiana State University,
Baton Rouge, Louisiana

MacroSoft, *Warsaw, Poland*

Marquette University,
Department of Mathematics,
Statistics and Computer Science,
Milwaukee, Wisconsin

Masaryk University,
Brno, Czechoslovakia

Mathematical Reviews,
American Mathematical Society,
Ann Arbor, Michigan

Max Planck Institut
für Mathematik,
Bonn, Federal Republic of Germany

NASA Goddard
Space Flight Center,
Greenbelt, Maryland

National Institutes of Health,
Bethesda, Maryland

National Research Council
Canada, Computation Centre,
Ottawa, Ontario, Canada

Naval Postgraduate School,
Monterey, California

New York University,
Academic Computing Facility,
New York, New York

Nippon Telegraph &
Telephone Corporation,
Software Laboratories,
Tokyo, Japan

Northrop Corporation,
Palos Verdes, California

Observatoire de Genève,
Université de Genève,
Sauverny, Switzerland

The Open University,
Academic Computing Services,
Milton Keynes, England

Pennsylvania State University,
Computation Center,
University Park, Pennsylvania

Personal TEX, Incorporated,
Mill Valley, California

Politecnico di Torino,
Torino, Italy

Princeton University,
Princeton, New Jersey

Purdue University,
West Lafayette, Indiana

Queens College,
Flushing, New York

Rice University,
Department of Computer Science,
Houston, Texas

Roanoke College,
Salem, VA

Rogaland University,
Stavanger, Norway

Ruhr Universität Bochum,
Rechenzentrum,
Bochum, Federal Republic of
Germany

Rutgers University, Hill Center,
Piscataway, New Jersey

St. Albans School,
Mount St. Alban, Washington,
D.C.

Smithsonian Astrophysical
Observatory, Computation Facility,
Cambridge, Massachusetts

Software Research Associates,
Tokyo, Japan

Space Telescope Science Institute,
Baltimore, Maryland

Springer-Verlag,
Heidelberg, Federal Republic of Germany

Springer-Verlag New York, Inc.,
New York, New York

Stanford Linear
Accelerator Center (SLAC),
Stanford, California

Stanford University,
Computer Science Department,
Stanford, California

Talaris Systems, Inc.,
San Diego, California

Texas A & M University,
Department of Computer Science,
College Station, Texas

UNI-C, *Aarhus, Denmark*

United States Military Academy,
West Point, New York

University of Alabama,
Tuscaloosa, Alabama

University of British Columbia,
Computing Centre,
Vancouver, British Columbia, Canada

University of British Columbia,
Mathematics Department,
Vancouver, British Columbia, Canada

University of Calgary,
Calgary, Alberta, Canada

University of California, Berkeley,
Space Astrophysics Group,
Berkeley, California

University of California, Irvine,
Information & Computer Science,
Irvine, California

University of California,
Los Angeles, Computer
Science Department Archives,
Los Angeles, California

University of California, Santa
Barbara, *Santa Barbara, California*

University of Canterbury,
Christchurch, New Zealand

University College,
Cork, Ireland

University of Crete,
Institute of Computer Science,
Heraklio, Crete, Greece

University of Delaware,
Newark, Delaware

University of Exeter,
Computer Unit,
Exeter, Devon, England

University of Glasgow,
Department of Computing Science,
Glasgow, Scotland

University of Groningen,
Groningen, The Netherlands

University of Heidelberg,
Computing Center,
Heidelberg, Germany

University of Illinois at Chicago,
Computer Center,
Chicago, Illinois

University of Kansas,
Academic Computing Services,
Lawrence, Kansas

Universität Koblenz–Landau,
Koblenz, Federal Republic of Germany

University of Maryland,
Department of Computer Science,
College Park, Maryland

University of Massachusetts,
Amherst, Massachusetts

Università degli Studi di Trento,
Trento, Italy

University of Oslo,
Institute of Informatics,
Blindern, Oslo, Norway

University of Oslo,
Institute of Mathematics,
Blindern, Oslo, Norway

University of Salford,
Salford, England

University of Southern California,
Information Sciences Institute,
Marina del Rey, California

University of Stockholm,
Department of Mathematics,
Stockholm, Sweden

University of Texas at Austin,
Austin, Texas

University of Washington,
Department of Computer Science,
Seattle, Washington

University of Waterloo,
Library–Serials Department,
Waterloo, Ontario, Canada

University of Western Australia,
Regional Computing Centre,
Nedlands, Australia

Uppsala University,
Uppsala, Sweden

Villanova University,
Villanova, Pennsylvania

Virginia Polytechnic Institute,
Interdisciplinary Center
for Applied Mathematics,
Blacksburg, Virginia

Vrije Universiteit,
Amsterdam, The Netherlands

Washington State University,
Pullman, Washington

Widener University,
Computing Services,
Chester, Pennsylvania

Worcester Polytechnic Institute,
Worcester, Massachusetts

Yale University,
Department of Computer Science,
New Haven, Connecticut

Index of Advertisers

Individual Membership Application

TEX USERS GROUP

Complete and return this form with payment to:

TEX Users Group
Membership Department
P. O. Box 21041
Santa Barbara, CA 93121-1041
USA

Membership is effective from January 1 to December 31 and includes subscriptions to *TUGboat, The Communications of the TEX Users Group* and the TUG newsletter, *TEX and TUG News*. Members who join after January 1 will receive all issues published that calendar year.

For more information ...

Whether or not you join TUG now, feel free to return this form to request more information. Be sure to include your name and address in the spaces provided to the right.

Check all items you wish to receive below:

- ☐ Institutional membership information
- ☐ Course and meeting information
- ☐ Advertising rates
- ☐ More information on TEX

Correspondence unaccompanied by a payment should be directed to

TEX Users Group
P. O. Box 869
Santa Barbara, CA 93102 USA

Telephone: (805) 899-4673
Email: `tug@Math.AMS.org`

Name _____

Institutional affiliation, if any _____

Position _____

Address (business or home (circle one)) _____

_____

City _____

State or Country _____ Zip _____

Daytime telephone _____ FAX _____

Email addresses (*please specify networks, as well*) _____

_____

I am also a member of the following other TEX organizations:

Specific applications or reasons for interest in TEX:

Hardware on which TEX is used:

Computer and operating system *Output device/printer*

_____

_____

_____

There are two types of TUG members: regular members, who pay annual dues of $60; and full-time student members, whose annual dues are $30. Students must include verification of student status with their applications.

Please indicate the type of membership for which you are applying:

☐ Regular @ $60 ☐ Full-time student @ $30

Amount enclosed for 1993 membership: $ _____

(*Prepayment in US dollars drawn on a US bank is required*)

☐ Check/money order payable to TEX Users Group enclosed

☐ Charge to MasterCard/VISA

Card # _____ Exp. date _____

Signature _____

TEX USERS GROUP

Institutional Membership Application

Institution or Organization _____

_____

Principal contact _____

Address _____

_____

City _____

State or Country _____ Zip _____

Daytime telephone _____ FAX _____

Email addresses (*please specify networks, as well*) _____

_____

Complete and return this form with payment to:

TEX Users Group
Membership Department
P. O. Box 21041
Santa Barbara, CA 93121-1041
USA

Membership is effective from January 1 to December 31. Members who join after January 1 will receive all issues of *TUGboat* published that calendar year.

Each Institutional Membership entitles the institution to:

- designate a number of individuals to have full status as TUG individual members;
- take advantage of reduced rates for TUG meetings and courses for *all* staff members;
- be acknowledged in every issue of *TUGboat* published during the membership year.

Educational institutions receive a $100 discount in the membership fee. The three basic categories of Institutional Membership each include a certain number of individual memberships. Additional individual memberships may be obtained at the rates indicated. Fees are as follows:

Category	Rate (educ./non-educ.)	Add'l mem.
A (includes 7 memberships)	$ 540 / $ 640	$50 ea.
B (includes 12 memberships)	$ 815 / $ 915	$50 ea.
C (includes 30 memberships)	$1710 / $1810	$40 ea.

For more information ...

Correspondence
TEX Users Group
P. O. Box 869
Santa Barbara, CA 93102 USA

Telephone: (805) 899-4673
Email: `tug@math.ams.org`

Please indicate the type of membership for which you are applying:

Category ____ + ____ additional individual memberships

Amount enclosed for 1993 membership: $ _____

☐ Check/money order payable to TEX Users Group enclosed

(*payment is required in US dollars drawn on a US bank*)

☐ Charge to MasterCard/VISA

Card # _____ Exp. date _____

Signature _____

Whether or not you join TUG now, feel free to return this form to request more information.

Check all items you wish to receive below:

☐ Course and meeting information

☐ Advertising rates

☐ More information on TEX

Please attach a corresponding list of individuals whom you wish to designate as TUG individual members. Minimally, we require names and addresses so that TUG publications may be sent directly to these individuals, but we would also appreciate receiving the supplemental information regarding phone numbers, email addresses, TEX interests, and hardware configurations as requested on the TUG Individual Membership Application form. For this purpose, the latter application form may be photocopied and mailed with this form.

TEX *without* Bitmaps

Wouldn't it be nice to be able to preview DVI files at any magnification, not just those for which bitmap fonts have been pre-built? Or to produce truly resolution-independent output that will run on any PostScript device, whether image setter or laser printer?

Perhaps you are looking for an alternative to Computer Modern? There now exist complete outline font sets which include math fonts that are direct replacements for those in CM. Even if you do want to remain faithful to CM, there are distinct advantages to switching to the outline version of the fonts. We supply the tools to do all of this:

DVIWindo — *preview DVI files calling for outline fonts*

* Preview at arbitrary magnification
* Preview in MS Windows™ — a simple, standardized user interface
* Print to any printer with a Windows printer driver
* Show EPSF files with preview on screen — and insert TIFF images

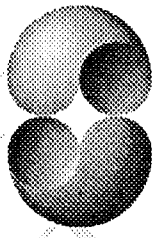
DVIPSONE — *partial font downloading for speed and efficiency*

* Avoid running out of memory on the printer
* Produce truly resolution-independent and page-independent output
* Designed from the bottom up for use with outline fonts on the PC

Fonts — *available from Y&Y in Adobe Type 1™ form (ATM compatible)*

* BSR Computer Modern fonts — with accented characters built in
* LaTeX + SLiTeX font set — line, circle, symbol, lcmss*, and logo*
* AMS font set — Euler, math symbol and Cyrillic fonts
* Lucida® Bright + Lucida New Math — a complete alternative to CM

Resolution-independent PostScript files using outline fonts can be printed by any service bureau, not just those with TEXpertise — and that translates into considerable savings for you. Is it perhaps time to get rid of those huge, complex directories full of bitmap fonts?



Y&Y, 106 Indian Hill, Carlisle, MA 01741 — (800) 742-4059 — (508) 371-3286 — Fax: (508) 371-2004

AP-TEX Fonts

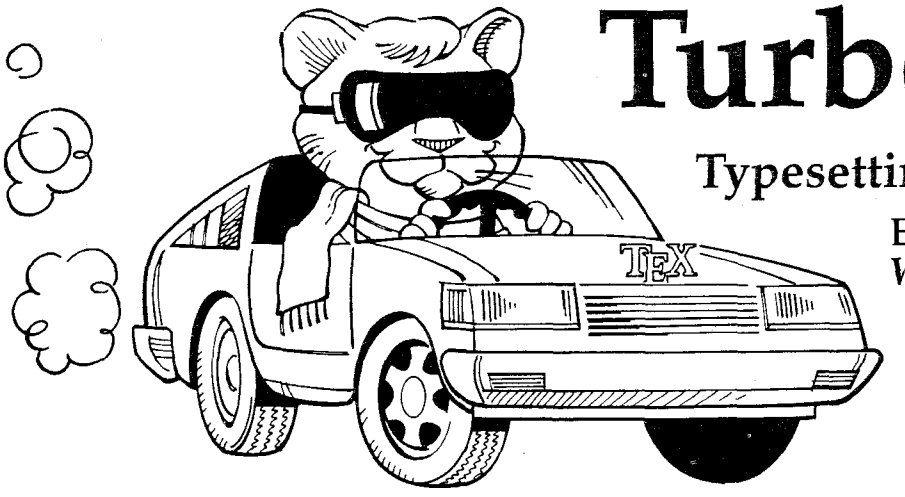
TEX-compatible Bit-Mapped Fonts Identical to Adobe PostScript Typefaces

If you are hungry for new TEX fonts, here is a feast guaranteed to satisfy the biggest appetite! The AP-TEX fonts serve you a banquet of gourmet delights: 438 fonts covering 18 sizes of 35 styles, at a total price of $200. The AP-TEX fonts consist of PK and TFM files which are exact TEX-compatible equivalents (including "hinted" pixels) to the popular PostScript name-brand fonts shown at the right. Since they are directly compatible with any standard TEX implementation (including kerning and ligatures), you don't have to be a TEX expert to install or use them.

When ordering, specify resolution of 300 dpi (for laser printers), 180 dpi (for 24-pin dot matrix printers), or 118 dpi (for previewers). Each set is on ten 360 KB 5-1/4" PC floppy disks. The $200 price applies to the first set you order; order additional sets at other resolutions for $60 each. A 30-page user's guide fully explains how to install and use the fonts. Sizes included are 5, 6, 7, 8, 9, 10, 11, 12, 14.4, 17.3, 20.7, and 24.9 points; headline styles (equivalent to Times Roman, Helvetica, and Palatino, all in bold) also include sizes 29.9, 35.8, 43.0, 51.6, 61.9, and 74.3 points.

The Kinch Computer Company
PUBLISHERS OF TURBOTEX
501 South Meadow Street
Ithaca, New York 14850
Telephone (607) 273-0222
FAX (607) 273-0484

Avant Garde Bold
Avant Garde Bold Oblique
Avant Garde Demibold
Avant Garde Demibold Oblique
Bookman Light
Bookman Light Italic
Bookman Demibold
Bookman Demibold Italic
Courier
Courier Oblique
Courier Bold
Courier Bold Oblique
Helvetica
Helvetica Oblique
Helvetica Bold
Helvetica Bold Oblique
Helvetica Narrow
Helvetica Narrow Oblique
Helvetica Narrow Bold
Helvetica Narrow Bold Oblique
Schoolbook New Century Roman
Schoolbook New Century Italic
Schoolbook New Century Bold
Schoolbook New Century Bold Italic
Palatino Roman
Palatino Italic
Palatino Bold
Palatino Bold Italic
Times Roman
Times Italic
Times Bold
Times Bold Italic
Zapf Chancery Medium Italic
Symbol ΔΦΓϑΛΠΘ
Zapf Dingbats ✂☞❑



TurboTEX

Typesetting Software

Executables $150
With Source $300

MICROSOFT® WINDOWS™ COMPATIBLE

THE MOST VERSATILE TEX ever published is breaking new ground in the powerful and convenient graphical environment of Microsoft Windows: TurboTEX Release 3.1E. TurboTEX runs on all the most popular operating systems (Windows, MS-DOS, OS/2, and UNIX) and provides the latest TEX 3.14 and METAFONT 2.7 standards and certifications: preloaded plain TEX, LATEX, AMS-TEX and AMS-LATEX, previewers for PC's and X-servers, METAFONT, Computer Modern and LATEX fonts, and printer drivers for HP LaserJet and DeskJet, PostScript, and Epson LQ and FX dot-matrix printers.

■ **Best-selling Value:** TurboTEX sets the world standard for power and value among TEX implementations: one price buys a complete, commercially-hardened typesetting system. *Computer* magazine recommended it as "the version of TEX to have," *IEEE Software* called it "industrial strength," and thousands of satisfied users around the globe agree.

TurboTEX gets you started quickly, installing itself automatically under MS-DOS or Microsoft Windows, and compiling itself automatically under UNIX. The 90-page User's Guide includes generous examples and a full index, and leads you step-by-step through installing and using TEX and METAFONT.

■ **Classic TEX for Windows.** Even if you have never used Windows on your PC, the speed and power of TurboTEX will convince you of the benefits. While the TEX command-line options and *TEXbook* interaction work the same, you also can control TEX using friendly icons, menus, and dialog boxes. Windows protected mode frees you from MS-DOS limitations like DOS extenders, overlay swapping, and scarce memory. You can run long TEX formatting or printing jobs in the background while using other programs in the foreground.

■ **MS-DOS Power, Too:** TurboTEX still includes the plain MS-DOS programs. Virtual memory simulation provides the same sized TEX that runs on multi-megabyte mainframes, with capacity for large documents, complicated formats, and demanding macro packages.

■ **Source Code:** The portable C source to TurboTEX consists of over 100,000 lines of generously commented TEX, TurboTEX, METAFONT, previewer, and printer driver source code, including: our WEB system in C; PASCHAL, our proprietary Pascal-to-C translator; Windows interface; and preloading, virtual memory, and graphics code, all meeting C portability standards like ANSI and K&R.

■ **Availability & Requirements:** TurboTEX executables for IBM PC's include the User's Guide and require 640K, hard disk, and MS-DOS 3.0 or later. Windows versions run on Microsoft Windows 3.0 or 3.1. Order source code (includes Programmer's Guide) for other machines. On the PC, source compiles with Microsoft C, Watcom C 8.0, or Borland C++ 2.0; other operating systems need a 32-bit C compiler supporting UNIX standard I/O. Specify 5-1/4" or 3-1/2" PC-format floppy disks.

■ **Upgrade at Low Cost.** If you have TurboTEX Release 3.0, upgrade to the latest version for just $40 (executables) or $80 (including source). Or, get either applicable upgrade free when you buy the AP-TEX fonts (see facing page) for $200!

■ ***No-risk trial offer:*** *Examine the* documentation and run the PC TurboTEX for 10 days. If you are not satisfied, return it for a 100% refund or credit. (Offer applies to PC executables only.)

■ **Free Buyer's Guide:** Ask for the *free, 70-page Buyer's Guide* for details on TurboTEX and dozens of TEX-related products: previewers, TEX-to-FAX and TEX-to-Ventura/Pagemaker translators, optional fonts, graphics editors, public domain TEX accessory software, books and reports.

Ordering TurboTEX

Ordering TurboTEX is easy and delivery is fast, by phone, FAX, or mail. Terms: Check with order (free media and ground shipping in US), VISA, Mastercard (free media, shipping extra); Net 30 to well-rated firms and public agencies (shipping and media extra). Discounts available for quantities or resale. International orders gladly expedited via Air or Express Mail.

The Kinch Computer Company
PUBLISHERS OF TURBOTEX
501 South Meadow Street
Ithaca, New York 14850 USA
Telephone (607) 273-0222
FAX (607) 273-0484

TEX CONSULTING & PRODUCTION SERVICES

North America

Abrahams, Paul

214 River Road, Deerfield, MA 01342; (413) 774-5500

Development of TEX macros and macro packages. Short courses in TEX. Editing assistance for authors of technical articles, particularly those whose native language is not English My background includes programming, computer science, mathematics, and authorship of *TEX for the Impatient*.

American Mathematical Society

P. O. Box 6248, Providence, RI 02940; (401) 455-4060

Typesetting from DVI files on an Autologic APS Micro-5 or an Agfa Compugraphic 9600 (PostScript). Times Roman and Computer Modern fonts. Composition services for mathematical and technical books and journal production.

Anagnostopoulos, Paul C.

433 Rutland Street, Carlisle, MA 01741; (508) 371-2316

Composition and typesetting of high-quality books and technical documents. Production using Computer Modern or any available PostScript fonts. Assistance with book design. I am a computer consultant with a Computer Science education.

ArborText, Inc.

1000 Victors Way, Suite 400, Ann Arbor, MI 48108; (313) 996-3566

TEX installation and applications support. TEX-related software products.

Archetype Publishing, Inc., Lori McWilliam Pickert

P. O. Box 6567, Champaign, IL 61821; (217) 359-8178

Experienced in producing and editing technical journals with TEX; complete book production from manuscript to camera-ready copy; TEX macro writing including complete macro packages; consulting.

The Bartlett Press, Inc., Frederick H. Bartlett

Harrison Towers, 6F, 575 Easton Avenue, Somerset, NJ 08873; (201) 745-9412

Vast experience: 100+ macro packages, over 30,000 pages published with our macros; over a decade's experience in all facets of publishing, both TEX and non-TEX; all services from copyediting and design to final mechanicals.

Cowan, Dr. Ray F.

141 Del Medio Ave. #134, Mountain View, CA 94040; (415) 949-4911

Ten Years of TEX and Related Software Consulting, Books, Documentation, Journals, and Newsletters. TEX & LATEX macropackages, graphics; PostScript language applications; device drivers; fonts; systems.

Electronic Technical Publishing Services Co.

2906 Northeast Glisan Street, Portland, Oregon 97232-3295;

(503) 234-5522; FAX: (503) 234-5604

Total concept services include editorial, design, illustration, project management, composition and prepress. Our years of experience with TEX and other electronic tools have brought us the expertise to work effectively with publishers, editors, and authors. ETP supports the efforts of the TEX Users Group and the world-wide TEX community in the advancement of superior technical communications.

NAR Associates

817 Holly Drive E. Rt. 10, Annapolis, MD 21401; (410) 757-5724

Extensive long term experience in TEX book publishing with major publishers, working with authors or publishers to turn electronic copy into attractive books. We offer complete free lance production services, including design, copy editing, art sizing and layout, typesetting and repro production. We specialize in engineering, science, computers, computer graphics, aviation and medicine.

Ogawa, Arthur

1101 San Antonio Road, Suite 413, Mountain View, CA 94043-1002; (415) 691-1126; `ogawa@applelink.apple.com`.

Specialist in fine typography, LATEX book production systems, database publishing, and SGML. Programming services in TEX, LATEX, PostScript, SGML, DTDs, and general applications. Instruction in TEX, LATEX, and SGML. Custom fonts.

Pronk&Associates Inc.

1129 Leslie Street, Don Mills, Ontario, Canada M3C 2K5;

(416) 441-3760; Fax: (416) 441-9991

Complete design and production service. One, two and four-color books. Combine text, art and photography, then output directly to imposed film. Servicing the publishing community for ten years.

Quixote Digital Typography, Don Hosek

349 Springfield, #24, Claremont, CA 91711; (714) 621-1291

Complete line of TEX, LATEX, and METAFONT services including custom LATEX style files, complete book production from manuscript to camera-ready copy; custom font and logo design; installation of customized TEX environments; phone consulting service; database applications and more. Call for a free estimate.

Richert, Norman

1614 Loch Lake Drive, El Lago, TX 77586; (713) 326-2583

TEX macro consulting.

TEXnology, Inc., Amy Hendrickson

57 Longwood Ave., Brookline, MA 02146; (617) 738-8029

TEX macro writing (author of MacroTEX); custom macros to meet publisher's or designer's specifications; instruction.

Type 2000

16 Madrona Avenue, Mill Valley, CA 94941;

(415) 388-8873; FAX (415) 388-8865

$2.50 per page for 2000 DPI TEX camera ready output! We have a three year history of providing high quality and fast turnaround to dozens of publishers, journals, authors and consultants who use TEX. Computer Modern, Bitstream and METAFONT fonts available. We accept DVI files only and output on RC paper. $2.25 per page for 100+ pages, $2.00 per page for 500+ pages.

Outside North America

TypoTEX Ltd.

Electronical Publishing, Battyány u. 14. Budapest, Hungary H-1015; (036) 11152 337

Editing and typesetting technical journals and books with TEX from manuscript to camera ready copy. Macro writing, font designing, TEX consulting and teaching.

Information about these services can be obtained from:

TEX Users Group
P. O. Box 869
Santa Barbara, CA 93102
(805) 899-4673

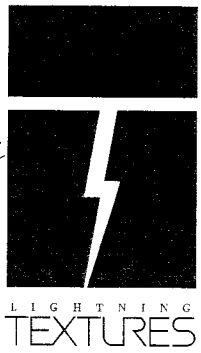
"Lightning Textures works wonders. This has saved me many hours of corrections...I have three books to be completed and this new gadget certainly gives me courage." anonymous graduate student, University of Ottawa

"I find that Lightning sets type on my Quadra 700 faster than the VAX in the Lab can do it. It's a very nice program, indeed." John C. Allred, Los Alamos National Laboratory

"I'll spread the word---Lightning Textures is great!" Prof. Anthony Siegman, Stanford University

"...a fantastic product and I can't imagine going back to an ordinary TEX environment." Chuck Bouldin, Naval Research Laboratory

"I used TEX on a PC for years, but now that I've used Textures I wouldn't (want to) go back to that for anything." George Killough, Hendecagon Corporation



"I just typeset 300 pages of two-column heavy math in 7.45 minutes on a Mac IIci -- almost a 4X speedup (actually 3.8). I am very impressed. Good job." Stanley Rabinowitz, Editor of Index to Mathematical Problems

"A few weeks ago, I received a copy of Lightning Textures from you. I was rather skeptical when I ordered it. I could hardly believe that Lightning would work the way you described...But, after testing the program, I am fully satisfied, even more, I am excited about Lightning: Congratulations!" Dr.Bernd Fischer, Universität Hamburg

"I have Textures installed and running -- -- it's great! Who could prefer other implementations of TEX, I wonder?." Mark Seymour, Springer-Verlag Heidelberg

"It's heaven." Ken Dreyhaupt, Springer-Verlag

"I know that Blue Sky Research did not intend Lighting Textures as a hackers's tool, but it sure can be used that way. It is strange how a simple speedup in the turnaround cycle (of about a factor of 10 to 100) changes your perspective. Lightning Textures is a joy to use." Victor Eijkhout, University of Tennessee

"I am absolutely amazed by Lightning Textures. It beats everything else hands down. I can't imagine what you could be putting in next." Kaveh Bazarghan, Focal Image

TUGBOAT

Volume 13, Number 4 / December 1992