

To them that already understand, more knowledge will be  
freely given. To the rest of us dolts, it's a struggle.

Jared M. Diamond  
The Ethnobiologist's Dilemma,  
*Natural History* (June 1989)

# TUGBOAT

COMMUNICATIONS OF THE T<sub>E</sub>X USERS GROUP  
EDITOR BARBARA BEETON

VOLUME 10, NUMBER 2                      •                      JULY 1989  
PROVIDENCE                      •                      RHODE ISLAND                      •                      U.S.A.

## TUGboat

During 1989, the communications of the T<sub>E</sub>X Users Group will be published in four issues. One issue will consist primarily of the Proceedings of the Annual Meeting.

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

Submissions to TUGboat are for the most part reproduced with minimal editing, and any questions regarding content or accuracy should be directed to the authors, with an information copy to the Editor.

## Submitting Items for Publication

The deadline for submitting items for Vol. 10, No. 3, is September 11, 1989; the issue will be mailed in November. (Deadlines for future issues are listed in the Calendar, page 285.)

Manuscripts should be submitted to a member of the TUGboat Editorial Committee. 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, in care of the TUG office.

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 T<sub>E</sub>X or L<sup>A</sup>T<sub>E</sub>X, will be sent on request; please specify which is preferred. For instructions, write or call Karen Butler at 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.

## TUGboat Advertising and Mailing Lists

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

## TUGboat Editorial Committee

Barbara Beeton, *Editor*

Ron Whitney, *Production Assistant*

Helmut Jürgensen, *Associate Editor, Software*

Laurie Mann, *Associate Editor, Training Issues*

Georgia K.M. Tobin, *Associate Editor, Font Forum*

Don Hosek, *Associate Editor, Output Devices*

Jackie Damrau, *Associate Editor, L<sup>A</sup>T<sub>E</sub>X*

Alan Hoenig and Mitch Pfeffer, *Associate Editors,*

*Typesetting on Personal Computers*

*See page 145 for addresses.*

## Other TUG Publications

TUG publishes the series T<sub>E</sub>Xniques, in which have appeared user manuals for macro packages and T<sub>E</sub>X-related software, as well as the Proceedings of the 1987 and 1988 Annual Meetings. Other publications on T<sub>E</sub>Xnical 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 T<sub>E</sub>X 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, contact Karen Butler at 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.

AMS-T<sub>E</sub>X is a trademark of the American Mathematical Society.

APS  $\mu$ 5 is a trademark of Autologic, Inc.

METAFONT is a trademark of Addison-Wesley Inc.

PC T<sub>E</sub>X is a registered trademark of Personal T<sub>E</sub>X, Inc.

PostScript is a trademark of Adobe Systems, Inc.

T<sub>E</sub>X is a trademark of the American Mathematical Society.

UNIX is a trademark of AT&T Bell Laboratories.

## Addresses

**Note:** Unless otherwise specified, network addresses (shown in typewriter font) are on the Internet.

**TeX Users Group Office**  
P. O. Box 9506  
Providence, RI 02940-9506  
or  
653 North Main Street  
Providence, RI 02904  
401-751-7760  
TUG@Math.AMS.com

**Peter Abbott**  
Computing Service  
Aston University  
Aston Triangle  
Birmingham B4 7ET, England  
21 359 5492  
pabbott@nsfnet-relay.ac.uk  
Janet: abbott@uk.ac.aston

**Phil Andrews**  
Pittsburgh Supercomputer Center  
Mellon Institute  
4400 Fifth Avenue  
Pittsburgh, PA 15213  
412-268-5006  
andrews%cpwsc@clipr.psc.edu

**Michael Ballantyne**  
TeXplorators  
3701 W. Alabama  
Suite 450-273  
Houston, TX 77027

**Stephan v. Bechtolsheim**  
2119 Old Oak Drive  
W Lafayette, IN 47906  
317-463-0162  
svb@cs.purdue.edu

**Barbara Beeton**  
American Mathematical Society  
P. O. Box 6248  
Providence, RI 02940  
401-272-9500  
bnb@Math.AMS.com  
TUGboat@Math.AMS.com

**Karen Butler**  
TeX Users Group  
P. O. Box 9506  
Providence, RI 02940-9506  
401-751-7760  
TUG@Math.AMS.com

**Lance Carnes**  
% Personal TeX  
12 Madrona Avenue  
Mill Valley, CA 94941  
415-388-8853

**S. Bart Childs**  
Dept of Computer Science  
Texas A & M University  
College Station, TX 77843-3112  
409-845-5470  
bart@cssun.tamu.edu  
Bitnet: Bart@TAMLSR

**Malcolm Clark**  
Imperial College Computer Centre  
Exhibition Road  
London SW7 2BP, England  
Janet: texline@uk.ac.ic.cc.vaxa

**John M. Crawford**  
Computing Services Center  
College of Business  
Ohio State University  
Columbus, OH 43210  
614-292-1741  
crawford-j@osu-20.ircc.ohio-state.edu  
Bitnet: CRAW4D@OHSTVMA

**Jackie Damrau**  
Mission Research Corporation  
1720 Randolph Road SE  
Albuquerque, NM 87106-4245  
505-768-7647  
damrau@dbitch.unm.edu  
Bitnet: damrau@bootes

**Michael DeCorte**  
2300 Naudain St. "H"  
Philadelphia, PA 19146  
215-546-0497  
mrd@sun.soe.Clarkson.edu  
Bitnet: mrd@clutx

**Allen R. Dyer**  
13320 Tridelphia Road  
Ellicott City, MD 21043  
301-243-0008 or 243-7283

**Shawn Farrell**  
Computing Centre  
McGill University  
805 Sherbrooke St W  
Montréal H3A 2K6, Québec, Canada  
514-398-3676  
Bitnet: CCSF@MCGILLA

**Jim Fox**  
Academic Computing Center HG-45  
University of Washington  
3737 Brooklyn Ave NE  
Seattle, WA 98105  
206-543-4320  
fox@uwavm.acs.washington.edu  
Bitnet: fox7632@uwacdc

**David Fuchs**  
1775 Newell  
Palo Alto, CA 94303  
415-323-9436

**Richard Furuta**  
Department of Computer Science  
University of Maryland  
College Park, MD 20742  
301-454-1461  
furuta@mimsy.umd.edu

**Regina Girouard**  
American Mathematical Society  
P. O. Box 6248  
Providence, RI 02940  
401-272-9500 x224  
RMG@Math.AMS.com

**Raymond E. Goucher**  
TeX Users Group  
P. O. Box 9506  
Providence, RI 02940-9506  
401-751-7760  
REG@Math.AMS.com

**Dean Guenther**  
Computing Service Center  
Washington State University  
Pullman, WA 99164-1220  
509-335-0411  
Bitnet: Guenther@WSUVM1

**Hope Hamilton**  
National Center for  
Atmospheric Research  
P. O. Box 3000  
Boulder, CO 80307  
303-497-8915  
Hamilton@MMM.UCAR.Edu

**Brian Hamilton Kelly**  
School of Electrical Engineering &  
Science  
Royal Military College of Science  
Shrivenham  
Swindon SN6 8LA, England  
+44 (793) 785252  
Janet: rmcs-tex@uk.ac.cranfield

**Doug Henderson**  
Division of Library Automation  
Office of the President  
University of California  
300 Lakeside Drive, Floor 8  
Oakland, CA 94612-3550  
415-987-0561  
Bitnet: dlatex@ucbcmca

**Alan Hoenig**  
17 Bay Avenue  
Huntington, NY 11743  
516-385-0736

**Don Hosek**  
3916 Elmwood  
Stickney, IL 60402  
Bitnet: U33297@UICVM

**Patrick D. Ion**  
Mathematical Reviews  
416 Fourth Street  
P. O. Box 8604  
Ann Arbor, MI 48107  
313-996-5273  
ion@Math.AMS.com

**Helmut Jürgensen**  
Department of Computer Science  
University of Western Ontario  
London N6A 5B7, Ontario, Canada  
519-661-3560  
Bitnet: helmut@uwovax  
UUCP: helmut@julian

**David Kellerman**

Northlake Software  
812 SW Washington  
Portland, OR 97205  
503-228-3383

uucp: imagen!negami!davek

**Donald E. Knuth**

Department of Computer Science  
Stanford University  
Stanford, CA 94305  
DEK@Sail.Stanford.Edu

**David H. Kratzer**

Los Alamos National Laboratory  
P. O. Box 1663, C-10 MS B296  
Los Alamos, NM 87545  
(505) 667-2864  
dhk@lanl.gov

**Joachim Lammarsch**

Research Center of the University  
Heidelberg  
Im Neuenheimer Feld 293  
6900 Heidelberg 1  
West Germany  
Bitnet: RZ92@DHDURZ1

**Charlotte Laurendeau**

TeX Users Group  
P. O. Box 9506  
Providence, RI 02940-9506  
401-751-7760  
TUG@Math.AMS.com

**Yoke Lee**

(see Michael Ballantyne)

**Pierre A. MacKay**

Northwest Computer Support Group  
University of Washington  
Mail Stop DW-10  
Seattle, WA 98195  
206-543-6259; 545-2386  
MacKay@June.CS.Washington.edu

**Laurie Mann**

Stratus Computer  
55 Fairbanks Boulevard  
Marlboro, MA 01752  
617-460-2610  
uucp: harvard!anvil!es!Mann

**Robert W. McGaffey**

Martin Marietta Energy Systems, Inc.  
Building 9104-2  
P. O. Box Y  
Oak Ridge, TN 37831  
615-574-0618  
McGaffey%ORN.MFEnet@nmficc.arpa

**Frank Mittelbach**

Fachbereich Mathematik  
Universität Mainz  
Staudinger Weg 9  
D-6500 Mainz  
Federal Republic of Germany  
Bitnet: SCHOEPP@DMZNAT51

**Dezső Nagy**

Geological Survey of Canada  
1 Observatory Crescent  
Ottawa K1A 0Y3, Ontario, Canada  
613-995-5449

**David Ness**

803 Mill Creek Road  
Gladwyne, PA 19035  
215-649-3474

**M. Edward Nieland**

Systems Research Laboratories, Inc.  
2800 Indian Ripple Road  
Dayton, OH 45440-3696  
513-255-8846  
TNIELAND@AAMRL.AF.MIL

**Dr. Hubert Partl**

EDV-Zentrum  
Technische Universität Wien  
Wiedner Hauptstraße 8-10  
A-1040 Wien, Austria  
Bitnet: z3000pa@awituw01

**Mitch Pfeffer**

Suite 90  
148 Harbor View South  
Lawrence, NY 11559  
516-239-4110

**Craig Platt**

Department of Math & Astronomy  
Machray Hall  
University of Manitoba  
Winnipeg R3T 2N2, Manitoba, Canada  
204-474-9832  
CSnet: platt@uofm.cc.cdn  
Bitnet: platt@uofmcc

**Jon Radel**

P. O. Box 2276  
Reston, VA 22090

**Thomas J. Reid**

Computing Services Center  
Texas A&M University  
College Station, TX 77843  
409-845-8459

**Zalman Rubinstein**

University of Haifa  
Department of Mathematics and  
Computer Sciences  
Mount Carmel  
Haifa 31999 Israel

**David Salomon**

Computer Science Department  
School of Engineering and Computer  
Science  
California State University,  
Northridge  
18111 Nordhoff Street  
Northridge, CA 91330  
818-885-3398  
bccscdxs@csunb.csun.edu

**Rainer Schöpf**

Institut für Physik  
Johannes Gutenberg Universität  
D-6500 Mainz  
Federal Republic of Germany  
Bitnet: SCHOEPP@DMZNAT51

**Larry Sharlow**

10 Toltec #3  
Flagstaff, AZ 86001  
602-774-1630

**Michael Spivak**

TeXplorators  
3701 W. Alabama  
Suite 450-273  
Houston, TX 77027

**Christina Thiele**

Canadian Journal of Linguistics  
Carleton University  
Ottawa K1S 5B6, Ontario Canada  
Bitnet: WSSCAT@Carleton

**Georgia K.M. Tobin**

The Metafoundry  
OCLC Inc., MC 485  
6565 Frantz Road  
Dublin, OH 43017  
614-764-6087

**Andrew Trevorrow**

Kathleen Lumley College  
North Adelaide, SA, 5006, Australia  
(08) 267 1060  
ACSnet: atrevorrow@g.ua.oz

**Brother Eric Vogel, FSC**

P. O. Box 5150  
Saint Mary's College  
Moraga, CA 94575  
415-631-4296

**Samuel B. Whidden**

American Mathematical Society  
P. O. Box 6248  
Providence, RI 02940  
401-272-9500  
sbw@Math.AMS.com

**Ron Whitney**

TeX Users Group  
P. O. Box 9506  
Providence, RI 02940-9506  
rfw@Math.AMS.com  
TUGboat@Math.AMS.com

**Patricia P. Wilcox**

The Coolspring Banjo Works  
6617 Home Road  
Delaware, OH 43015  
614-881-5032  
banjo@DCI2PW.DAS.NET

**Hermann Zapf**

Seitersweg 35  
D-6100 Darmstadt  
Federal Republic of Germany

## General Delivery

### From the President

Bart Childs

The program of the annual meeting is set. The program committee has done an excellent job. It is appropriate to call this a celebration of 10 years of T<sub>E</sub>X and TUG.

I am writing this so soon after the distribution of the most recent TUGboat that I have not had any feedback on the **dingbat** competition. That announcement was missing a statement that a sample submission has been prepared by Doug Henderson and is available from TUG headquarters. I encourage you to submit a dingbat or two. Doug's sample dingbats (an anchor for TUG and a check mark) and methodology should be a big help. Send your entries to Doug; his address is on page 145. Judging will take place at the annual meeting.

See you at the celebration at Stanford.

has been informed of Score's demise, and they are investigating all reasonable possibilities for alternative locations. Several criteria are essential for a suitable new home: the machine must be on the Internet; it must support anonymous FTP; and a T<sub>E</sub>X installation must have the full support of the management. All news will be communicated via the electronic mailing lists—T<sub>E</sub>Xhax, UKT<sub>E</sub>X, T<sub>E</sub>XMAG, et al.—and published in TUGboat.

As long as Score is viable, updates to the T<sub>E</sub>X system will continue to be posted there. The current versions are

T<sub>E</sub>X 2.99  
 PLAIN.TeX 2.94  
 METAFONT 1.7  
 PLAIN.MF 1.7

Changes to T<sub>E</sub>X, METAFONT and the CM fonts can be obtained by requesting the latest errata list supplement from the TUG office.

All changes have been communicated to all the implementors and distributors on my mailing list. If you are creating a new implementation of T<sub>E</sub>X and distributing it to other users, you should be receiving this information. Send me your name and address (preferably an electronic address accessible via the Internet), and a short description of the implementation you're working on.

### Updates to items in TUGboat 10 #1, and a suggestion to authors

Several pairs of braces {...} were omitted from Georgia Tobin's article "A handy little font", making it nearly impossible for someone not already versed in METAFONT to recreate the fonts described there. Georgia's column in this issue contains the correct code for the handpointing character.

This brings up the subject of good coding practices. Some readers of TUGboat, I am told, take a new issue directly to their keyboard and enter the code to try it out for themselves. Authors of macros and of METAFONT code can make that activity easier for readers by making their code statements complete and precise. If redundant segments of code are omitted for conciseness, then that should be stated, along with some indication of how the missing code can be reconstructed; the techniques used in Appendix B of *The T<sub>E</sub>Xbook* may be a good model. Further, an author shouldn't assume that all readers are experts. For example, if @ is used in "internal" control sequences, appropriate \catcode'\@=. . . statements should be included in plain code, or the equivalent \makeat. . . in L<sup>A</sup>T<sub>E</sub>X.

A couple of years ago, a "Birds of a Feather" session at a TUG meeting discussed what are good

---

### Editorial Comments

Barbara Beeton

This column seems to be becoming my personal soap box, so I may as well make the most of it. I shall feel free to award compliments where due, and to nag gently when I think it is called for. Mostly, though, I shall try to bring to your attention items that I think are important or interesting, that haven't been mentioned anywhere else in the issue. This will usually include a report on what are the latest versions of the official T<sub>E</sub>X-related software, trip reports on meetings I've attended, and anything else that strikes my fancy. Suggestions from you readers are always welcome—the addresses for TUGboat, postal and electronic, are inside the front cover.

### T<sub>E</sub>X news

The big news about T<sub>E</sub>X, really about Stanford, is that the Score machine is to be unplugged on August 31, just after the TUG meeting. This computer has been home to the authoritative T<sub>E</sub>X distribution since the beginning. The TUG Board

macro coding practices. A number of good ideas were mentioned and recorded, but nothing has been published. I intend to encourage this work to begin again at the upcoming TUG meeting, and perhaps we will see some macro guidelines appear to join the emerging DVI driver standards. In the meantime, the suggestions by Don Hosek for creating portable METAFONT code (page 173) make sense for macros too.

### The TUGboat schedule

Unanticipated delays in the printing schedule for issue 10 #1 assured that it arrived too late for most potential authors to respond to the editorial deadline for #2. After considering various possibilities, we have decided to set fixed dates for future issues.

The first issue has always been scheduled to permit authors a break after the holiday rush. In the past, the second and third issue dates have depended on the schedule for the annual meeting; there was sentiment for issue #2 to be in members' hands in time to read before the meeting, and for the deadline for #3 to give sufficient time after the meeting to permit reactions by the authors to what happened there. (The fourth issue—the meeting proceedings—of course depends directly on the meeting schedule, but it has a separate editorial and production staff.)

The new schedule for the three regular issues will follow the same general pattern. For each issue, the editorial deadline will be a Tuesday, permitting authors a weekend for last-minute cleanup. Editorial and production work will be allowed six weeks (the Editor has a full-time job with the Math Society, and TUGboat is her evening and weekend activity). Camera copy will be delivered to the TUG office on a Monday for shipment to the printer. For the past year, the printer has required six weeks; the TUG office is investigating printers and expects to find a reliable one who can handle the job in four or even three weeks.

Editorial deadlines will be determined by the following.

Issue	
#1	3 <sup>rd</sup> Tuesday of January
#2	2 <sup>nd</sup> Tuesday of April
#3	Proceedings: as soon as possible, but no later than deadline for #4
#4	2 <sup>nd</sup> Tuesday after Labor Day

The long summer gap should accommodate a meeting scheduled any time from the middle of July through late August, and also give the Editor some time to relax.

The deadlines for 1990 are given in the calendar; see page 285. Applying the expected production, printing and mailing time, it can be seen that issue #1 may not reach prospective authors until after the deadline for issue #2. To provide an appropriate warning, it has been proposed that the schedule be sent along with acknowledgements for receipt of dues payments. We promise that deadlines will be shown in the calendar for at least two issues.

Informal discussion at the Congrès GUTenberg (see below) yielded some suggestions regarding the technical articles in TUGboat. These included peer review and actual testing of macros and METAFONT code. I believe it is time to consider these actions, as they would enhance the quality and utility of the presentations, but they will mean a change in the way that the proposed schedule is interpreted—the six weeks permitted for production simply doesn't allow the necessary time for communication between reviewer/editor/author should revisions be found desirable. Another suggestion was to include abstracts of the main articles in several languages (English and, say, French and German), also to accept articles in languages other than English. I will solicit opinions and report in the next issue.

So, what are your opinions? Let us hear from you, about the schedule, the suggestions concerning review and testing, or anything else about TUGboat. (If you've noticed no letters to the Editor in recent issues, you've only yourselves to blame.) Electronic mail is, as always, most convenient, and most likely to get a quick acknowledgement, but all comments will be considered carefully.

### TUGboat selections on-line

The following additions have been made to the directory <TeX.TUGBOAT> at Score.Stanford.edu since the last issue.

- the tables of contents, file TB1089.CNT, for this year's issues so far.
- FIGPLACE.TeX and FIGSPACE.TeX by Joost Zalmstra and David F. Rogers (TUGboat 10#1)

Check the file -CHRONO-.DIR for a chronological list of the directory contents, and -READ-.TUG and TUGFIL.CHG for a description of the files in the directory and details of changes.

These files are available through August 31 from Score via anonymous FTP on the Internet. Copies have also been installed in the archives at Clarkson and Aston. We are looking into ways to make them available from the TUG office for those who have no network access.

### Congrès GUTenberg — A trip report

The second annual Congrès GUTenberg was held in Paris on May 16–17, on the theme “ $\text{T}_{\text{E}}\text{X}$  and graphics”. I was going to prepare a detailed report of the program, but Malcolm Clark has done such a good job (see page 150) that I will limit this report to my impressions.

The first day of the Congrès was occupied by two short courses, the first an introduction to  $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ , by Olivier Nicole and Jacques André, and the second “first steps in METAFONT”, by Victor Ostromoukhov, which I attended. I had seen some results of Victor’s work in Exeter at  $\text{T}_{\text{E}}\text{X}88$ , and was pleased to be introduced to some of the basic techniques he uses to make character shapes look their best on a low-resolution display device while enforcing stylistic uniformity within a group of related characters. The letterforms he chose for demonstration were from the outline alphabet known as “blackboard bold”:  $\mathbf{R}$ ,  $\mathbf{Q}$ ,  $\mathbf{N}$ ,  $\mathbf{K}$ ,  $\mathbf{Z}$ ,  $\mathbf{H}$ ,  $\mathbf{F}$  and  $\mathbf{C}$ . (These samples are from the AMS font `msym10`, not from Victor’s production; I feel that he has corrected some of the problems known to exist in these old versions.) These were sufficient to illustrate such principles as uniformity of stem widths and line thickness, keeping paired lines parallel to one another when they are not parallel to an axis, and the like. Victor’s use of a Macintosh to demonstrate, almost instantaneously, both bad and good effects was a powerful reinforcement.

The official first day ended with the annual Assemblée Générale, where the order of the day was the presentation of secretary’s and treasurer’s reports, ratification of actions of the Board, and other items of GUTenberg business. Bernard Gaulle, GUTenberg’s president, presided. I learned that GUTenberg now has about 90 members and another 50 or so subscribers to their journal, the *Cahiers GUTenberg*. If my memory serves, TUG was about that size at the same age; at the age of 10, TUG has over 3,000 members, and I wish GUTenberg the same success. Thanks were extended to several organizations and persons who had contributed facilities or efforts toward strengthening GUTenberg or improving communications among  $\text{T}_{\text{E}}\text{X}$  users in the French-speaking world and elsewhere. These included the IRISA and CIRCE (and another, whose name I did not catch) laboratories, the École Normale Supérieure (rue d’Ulm), Peter Abbott (moderator of  $\text{UK}_{\text{E}}\text{X}$ ), and Pierre MacKay (moderator of  $\text{T}_{\text{E}}\text{X}hax$ ). Two individuals were made “membres d’honneur”. Raymond Seroul, author of *Le petit Livre de  $\text{T}_{\text{E}}\text{X}$*  was one, and I was very much

honored to be the other. As a token of the honor, I was presented with a copy of *Les bons Romains*, bound from semiweekly editions of 1870–1871 containing the complete *Le Comte de Monte Cristo* by Alexandre Dumas and selections by other authors including de Balzac; when I have finished reading it all, my understanding of French should be very much improved.

The second day, the conference proper, was opened with an introduction by the chairman of the session, Nicolas Brouard. This was followed by greetings from Bernard Gaulle and his recitation of all the anniversaries to be celebrated this year, starting with the bicentennial of the French Revolution. He reported on the concerns of the French  $\text{T}_{\text{E}}\text{X}$  community, the various sources of available information, and ended with a review of the business meeting of the previous evening. The presentation of papers then occupied the rest of the day.

Although I am aware that the desire and need for graphics inclusion in  $\text{T}_{\text{E}}\text{X}$  documents is very great in many environments, the fact that it is not the most urgent problem facing the Math Society has insulated me from most such activity. The scope and ingenuity of the approaches described by the speakers was most impressive. Many tools are available to assist in document preparation, and of these, quite a few are compatible with  $\text{T}_{\text{E}}\text{X}$ , though the graphics tools are usually dependent on the facilities of particular output devices. Nelson Beebe observed that the basic problem is that  $\text{T}_{\text{E}}\text{X}$  came ten years too soon. (Of course, if it hadn’t, we wouldn’t be able to celebrate ten years of  $\text{T}_{\text{E}}\text{X}$  this summer, and the Congrès GUTenberg wouldn’t have taken place.) PostScript was mentioned in a number of contexts, with or without connection to the Macintosh. Another topic that kept surfacing was that of standards, graphics and otherwise. Standards seem to have generally stronger support in Europe than in the U.S., although with the adoption of SGML by the Department of Defense the awareness of standards has been raised considerably. The interoperability of  $\text{T}_{\text{E}}\text{X}$  and various document and graphics standards seems a very fruitful area for investigation. Several papers described more individualistic solutions to particular problems; the use of the “screen graphics” symbols to produce diagrams was one such. Another suggested approach was to reprocess DVI files to insert commands suitable for particular print engines. DVI portability is sometimes hard to realize even with text, and the problems are magnified with graphics. One goal that seems worth working for is the ability to archive documents for long-term storage; if graphics are to

be an integral part of such archived documents, then we might be warned to think hard before revising  $\TeX$  too soon.

I have already mentioned Raymond Seroul and his book, *Le petit Livre du  $\TeX$* . I would like to recommend this book highly (and not just because Raymond gave me a copy). It contains, among other useful features, a fine "Dictionnaire-Index", what I would call a glossary, which lists, with extensive explanations and references to the main text, all the important control sequences and other useful concepts, e.g. "acolade" (a brace), "fonctions (noms de...)". I have heard this kind of reference suggested many times, but *Le petit Livre* has the first example I have seen in print. TUG is investigating having it translated and making it available in English. In the meantime, the French edition should present little difficulty, and much information, to a reader who has some familiarity with the French language, and a dictionary.

Raymond Seroul,  
*Le petit Livre de  $\TeX$* ,  
 Préface de Dominique Foata.  
 InterEditions, Paris, 1989. 317 pp.  
 ISBN 2-7296-0233-X

Finally, I can also recommend the *Cahiers GUTenberg*, which is a window on the  $\TeX$  world with quite a different view than TUGboat. Malcolm Clark's report on the Congrès gives all the details of how to join GUTenberg and subscribe to the *Cahiers*.

---

### Réflexions sur le Congrès GUTenberg Paris, Mai 16-17, 1989

Malcolm Clark

The French  $\TeX$  users group has been around for a few years, in an 'unofficial' form. Over the last year or so they have become 'official', much more active and they now organise an annual meeting. The first well-publicised GUTenberg meeting was held in Paris last year. I was impressed there by the attendance (well over 100), the stamina (the room was tiny and without air-conditioning), and the range of topics covered. There is, in any case, a tradition of  $\TeX$  activity in France — the second European  $\TeX$  Conference was held in Strasbourg, in 1986.

This year's meeting was again in Paris, but used far larger rooms to accommodate the 120-150 people who attended. Scanning down the list of attendees, there are the usual academic and research organisations, but also publishers, and the printing trade in general. This bodes well for the future. The meeting was held over two days: the first day was given over to two seminars — one on  $\LaTeX$  (from Olivier Nicole and Jacques André), and the other on METAFONT (Victor Ostromoukhov); followed by the AGM. The second day was the conference proper. I attended part of Victor's 'Premiers pas en METAFONT'. As usual, I was impressed by Victor's breadth and depth of METAFONT-lore. And he seemed to be getting something useful across to the forty or so would be METAFONTers. Since he used a Macintosh to demonstrate the points, there was a reasonably quick interaction between intention and realisation. I confess I didn't stay to all of this; my powers of concentration are not great enough to follow a full day of technical METAFONT (far less in a foreign tongue). However, one quote from Victor:

```
<mathematical typesetting> → <empty>|< $\TeX$ >
< $\TeX$ >→< $\TeX$ 82><font support>
```

That sums things up nicely I think.

Bernard Gaulle, GUTenberg's President ran the AGM with great efficiency and some humour. I particularly like the French style of democracy (it runs: question — 'anyone against?'; answer — 'no'; conclusion — 'passed'; excellent). I won't plough through all the bits of the AGM, except to note that the group is in excellent financial health, that this is a year of anniversaries in France — 200 years since the Revolution, but also some others: 10 years of TUG, 50 years of CNRS (Centre Nationale de la Recherche Scientifique), one of the homes of  $\TeX$ , and of course this is also the year of the 4th European  $\TeX$  Conference. The AGM honoured two people with 'honorary membership': Barbara Beeton and Raymond Seroul (the author of *Le petit Livre de  $\TeX$* ). By way of recognition, they were each presented with an edition of 'Les bons Romains', published over a hundred years ago. Peter Abbott was also thanked for the help he has given in easing '[les] perturbations EARN/Bitnet'. I was particularly pleased to see Barbara honoured and Peter thanked in this way. The whole  $\TeX$  community owes them much for their dedicated adherence to the cause, and it was particularly refreshing and tactful that GUTenberg saw fit to include them in this way.

Besides this conference, GUTenberg produces its own journal, *Cahiers GUTenberg*. The inaugural (or prototype) edition (confusingly numbered 'zéro') was available at last year's conference. The first and



second editions were out by this year's conference. Many of the talks in the conference were also printed in the *Cahiers* (which helped me enormously). Although the group is 'francophone', several articles are in English. Allowing for the technical words which dictionaries never seem to get right, it isn't too difficult to make sense out of the papers/articles. The *Cahiers* represent a major undertaking, in time, effort, and in financial commitment (as I well know from my own limited venture in *TeXline*). If the high standards already being established are maintained, GUTenberg will have created something which will be of great and lasting service to the whole *TeX* community. My only minor criticism of the *Cahiers* is the lack of consistency in the provision of abstracts (a failing it shares with TUGboat). If abstracts were included, it would be possible to prepare multilingual translations which could be circulated more widely, alerting others to the range and relevance of the material.

The major theme of the Conference was 'graphics' — a popular one these days. Fortunately, thanks to Sebastian Rahtz' talks, I think I know a little about the background here. Rather than report each talk in detail, it is perhaps more productive to try to select some of the major themes. After all, the text of most of the talks is available. As usual, it is notable how far *L<sup>A</sup>T<sub>E</sub>X* dominates in Europe (or perhaps, just outside the US). It is also notable that 'standards', however defined, keep cropping up: X-Windows, POSTSCRIPT, *TeX* itself, PHIGS, GKS, SGML, and even emacs. This seems particularly healthy, although equally there are many forays into areas which are less portable. Nevertheless, the apparent domination of C, as the implementation language of choice, (with or without the spectre of Unix) would indicate the possibility of transferring some of the applications to other platforms. From the summaries, it is evident that one of the great concerns is the use of POSTSCRIPT, and the incorporation of POSTSCRIPT (and EPSF) files into (especially) *L<sup>A</sup>T<sub>E</sub>X*. Perhaps my favourite paper was Maurice Laugier's. His was a very simple and straightforward idea — namely that the PC's graphics characters may be mapped quite easily into rules, and that tables (and some diagrams) may be prepared by this means, provided that a monospaced font is adequate.

A list of the talks, together with a brief summary (usually the authors'/author's own) is included here. Some of the talks were in English (a bold move for a francophone group), and some summaries (notably that of Lance Carnes) were distributed in English and French. Very tactful.

*L'année de tous les anniversaires*: Bernard Gaulle. A welcome to GUTenberg, and an overview of the services and facilities available to *TeX* users in France (and elsewhere); touches on the public, private and commercial domains; addresses GUTenberg's relationship with the rest of the world.

*Xwindows, L<sup>A</sup>T<sub>E</sub>X, T<sub>E</sub>Xdraw et Plot79, ou comment calculer, rédiger, dessiner et imprimer plus aisément*: Nicolas Brouard. A workstation running under X-Windows offers a way of calculating, writing and designing which is much simpler than with a 'classic' terminal. Two graphics tools, *TeXDraw* (a public analogue of MacDraw) and *PLOT79* (a 3D graphics system based on CORE), can easily be employed with *L<sup>A</sup>T<sub>E</sub>X*.

*TeX and Graphics: the state of the problem*: Nelson Beebe. Inclusion of graphics in documents typeset by *TeX* is not yet a satisfactorily solved problem, and no final general solution is in sight. This paper surveys alternatives for insertion of graphics in *TeX* documents. It summarizes graphics primitives of several modern software systems, and shows how *TeX* has seriously deficient support for their direct incorporation in *TeX* itself.

*L'environnement de production de documents T<sub>E</sub>X à l'IRISA*: Philippe Louarn & Bertrand Decouty. The group consists of over 200 researchers, teachers and engineers, who need to produce reports, articles, theses, books... The objective was to provide a set of homogenous tools in a heterogenous environment (Unix, VMS, workstations, PCs...). Naturally, *TeX* was chosen as the fundamental tool. In graphics, the two main avenues are through the incorporation of POSTSCRIPT, and by the use of eepic in the *L<sup>A</sup>T<sub>E</sub>X* picture environment.

*Survey of T<sub>E</sub>X Graphics for the PC*: Lance Carnes. This talk reviews the various graphics systems offerings for the IBM PC and compatibles, and the ways these systems can be used in conjunction with *TeX*; the use of POSTSCRIPT, HP PCL and bitmap files in conjunction with current *TeX* drivers; the use of output from screen oriented drawing systems; and the conversion of graphics files from one format to another, and scaling of images, for inclusion in documents.

*TeX et les graphiques dans l'environnement Mac*: Anestis Antoniadis. Painting and drawing are the two sides of the creation of graphics on the Macintosh. *Paint* images (otherwise known as bitmaps) are known to the Mac as a set of points on the screen. *Draw* images (also known as

vector drawings) are known to the Mac as objects (rectangles, lines, circles, polygons) and are defined by their mathematical attributes. As a consequence of the way in which they are defined, they take full advantage of the resolution of POSTSCRIPT peripheral devices. The goal of this article is to give an overview of the methods and software for the generation of graphics on the Mac, and to discuss the insertion of such graphics in documents prepared with *Textures*, one of the implementations of T<sub>E</sub>X on the Macintosh.

**METAFONT et POSTSCRIPT:** Victor Ostromoukhov. Conversion between METAFONT and POSTSCRIPT is possible. What are the best techniques, and what are the constraints? (No written contribution, but see his MacMETAFONT program.)

**DDI: un environnement de travail pour la réalisation de graphiques scientifiques, techniques et fantaisies utilisables avec T<sub>E</sub>X:** André Violante. DDI is a work environment for the creation of scientific, technical and artistic graphics. The fundamental idea of the system is the creation and use of graphic fonts. To use these, several tools are available: design software (*Designcad*); a program to convert *Designcad* files to METAFONT; METAFONT itself; GFtoPK; T<sub>E</sub>X; and a suitable device driver.

**texpic: design and implementation of a picture graphics language in T<sub>E</sub>X à la pic:** Rolf Olejniczak. **texpic** is a T<sub>E</sub>X implementation of a graphics language similar to Kerhighan's troff preprocessor pic. The implementation consists of two parts, a set of elaborate T<sub>E</sub>X macros and a postprocessor for drawing (in the dvi file). **texpic** objects and T<sub>E</sub>X or L<sup>A</sup>T<sub>E</sub>X commands may be combined at will. **texpic** is written in C, and is fully portable, to the extent that every T<sub>E</sub>X implementation, every preview and every correctly written printer driver will work with **texpic**.

**Traduction en T<sub>E</sub>X d'un fichier SGML avec récupération des graphiques et des tableaux:** Maurice Laugier. The logic of SGML markup is close to that of L<sup>A</sup>T<sub>E</sub>X, and translation from SGML to L<sup>A</sup>T<sub>E</sub>X can be realised quite simply. However, the problem is rather different for graphics and tables. This paper presents a way in which the PC graphics characters were successfully mapped into L<sup>A</sup>T<sub>E</sub>X, easing the production of tables and simple diagrams.

**GIT<sub>E</sub>X, PAPS: deux logiciels manipulant POSTSCRIPT et L<sup>A</sup>T<sub>E</sub>X:** Christophe Cérin. From PC-based graphics programs, PAPS (Programme

d'Application POSTSCRIPT) transforms an image into a POSTSCRIPT format. It will also allow some manipulation of the graphics image. GIT<sub>E</sub>X (Générateur d'Image T<sub>E</sub>X) is a program which allows a figure environment to be constructed for inclusion of the POSTSCRIPT into a L<sup>A</sup>T<sub>E</sub>X file.

**L'incorporation de graphiques dans INRST<sub>E</sub>X:**

Michael Ferguson. The approach used in INRST<sub>E</sub>X is to use the power of the printer to create graphics, and not to introduce special characters. The capability to generate graphics has been achieved for POSTSCRIPT and for QMS graphics on a QUIC laser printer. The paper discusses the role of the printer as well as the need for support software to permit the incorporation of graphics produced by other systems. The paper also discusses some of the limitations inherent in the choice of graphics systems.

Like most conferences, the most interesting and valuable discussions take place in the corridors, over coffee, or at lunch. This tradition was maintained here. The corridors were also used to display various pieces of T<sub>E</sub>Xware. The inclusion of a noticeboard for general T<sub>E</sub>X-notice, trivial as it seems, was extremely useful. There was a display of ArborText's Publisher (about the only place you don't see Publisher these days is in the UK), and an extensive display of books, where Raymond Seroul's book was selling well (on its first day of publication).

## Conclusion

I was impressed. There can be no doubt about it, GUTenberg provides us all with much to emulate. The strength and coherence of the group is manifest. Taken purely at the national (or francophone) level, GUTenberg is contributing massively to the strength of T<sub>E</sub>X. The *Cahiers* are excellent, and look capable of sustained quality. The annual meeting is now well-established and imaginative. GUTenberg's involvement with several French publishers must also be a good sign.

GUTenberg is also keenly aware of other French speaking areas (Belgium, Switzerland, Quebec) and of the advantages of international electronic communication. And the committee ensured that the various national representatives (myself, representing TUG and UKTUG; Joachim Lammarsch, representing the German group, Dante; and Kees van der Laan representing the Dutch group) had the opportunity to discuss how we could cooperate for the common good.

I am a shade worried by what I see as the determination to stay outside the TUG orbit. Perhaps I am over-sensitive — being described either as English or Anglo-Saxon does tend to make me a trifle testy — but I have always thought of TUG as an international organisation, not an American one. Others do not share this perception. If our conclusion is that we are not getting what we need out of TUG, the solution is in our own hands. We can influence the organisations in which we participate. That's the key — participation. T<sub>E</sub>X must be worth it.

### Joining GUTenberg; subscribing to the Cahiers

To join GUTenberg, you need only part with 200 FF. This has two advantages (besides preparing you for 1992 and demonstrating your adherence to the European ideal) — it enables you to pay a reduced fee at the annual meeting and for the *Cahiers*. To obtain the *Cahiers* costs a further 150 FF if you are a member, but 250 FF if you are not. This year's conference cost 200 FF for members and 400 FF for non-members. Clearly membership pays for itself if you are contemplating attending the conference and taking the journal. In other words, joining GUTenberg and subscribing to the *Cahiers* costs you a total of 350 FF (made payable to GUTenberg). Of course Eurocheques are acceptable.

Note that membership comes in several different categories: individual membership is 200 FF; institutional membership on behalf of a non-profit organisation is 700 FF; while institutional membership on behalf of profit-making (as opposed to *profitable?*) organisation is a hefty 1400 FF. On the other hand, institutional membership does allow you to nominate up to seven individuals.

Send your money to:

GUTenberg

% IRISA

Campus Universitaire de Beaulieu

35042 Rennes Cedex

France

◇ Malcolm Clark  
Imperial College Computer Centre  
Exhibition Road  
London SW7 2BP, England  
Janet: `texline@uk.ac.ic.cc.vaxa`

---

## International Standards and T<sub>E</sub>X

Malcolm W Clark

Editor's note: The following "article" is really two papers that Malcolm prepared as notes for his presentation on standards at the June Nordic T<sub>E</sub>X meeting (see page 287). International standards are becoming increasingly important in technical publishing, and they will undoubtedly affect the way in which many T<sub>E</sub>X users carry on their work. It seems only fair that we should work to make T<sub>E</sub>X affect the way in which standards are defined. With Malcolm's permission, these papers have also been submitted to the U.S. working groups on the Office Document Architecture (ODA) standard, X3V1.3 and .5.

### Standards and T<sub>E</sub>X

T<sub>E</sub>X is a standard. It is rigidly defined; every implementation of T<sub>E</sub>X must pass the 'trip' test before it has the right to call itself 'T<sub>E</sub>X'. Even the components of T<sub>E</sub>X are standards; dvi format is defined rigorously; even the formats of px1, pk, and gf are defined. Thanks to this standardisation, every user of T<sub>E</sub>X (L<sup>A</sup>T<sub>E</sub>X and A<sub>M</sub>S-T<sub>E</sub>X) knows that he or she can expect the same results from the same input, no matter what equipment they use to prepare their document. This is a degree of standardisation which is unheard of outside the T<sub>E</sub>X world.

There are areas within the T<sub>E</sub>X world without adequate standards. Some of these have been at least considered by the T<sub>E</sub>X community. Both device driver standards and macro-writing standards have been the subject of working parties: the driver standards working party has made some preliminary announcements, but the macro-writing standards working party seems to be moribund. Other areas require consideration, including the handling of *specials*, but this may be tackled by the driver standards people.

But there are wider standards which affect us in the T<sub>E</sub>X world. The T<sub>E</sub>X world is only a small (and many would argue, privileged) part of the 'document' world. There are international and *de facto* standards which are of critical importance to us.

We have to acknowledge the importance of the *de facto* standard, POSTSCRIPT. In a sense, T<sub>E</sub>X 'sits above' POSTSCRIPT. No-one (well, almost no-one), would ever dream of writing a document in straight POSTSCRIPT. Normally, we write a program which generates POSTSCRIPT — for example,

**TeX**. Nevertheless, we must be aware of the way in which **POSTSCRIPT**-compatibility is crucial if we are to be taken seriously by the rest of the world. We must have an acceptable answer to the question 'Can you generate **POSTSCRIPT**?', even if we feel that the question is ill-posed.

The first 'international' standard which is of importance to us is **SGML**, the Standard Generalized Markup Language. **TeX** is itself a markup language, but **SGML** takes this one step further to become divorced completely from the realm of typesetting. **L<sup>A</sup>TeX** has a closer affinity to **SGML**, although it does not go quite far enough. While it is possible to argue that the basic paradigm of **SGML** is flawed, its widespread acceptance and use (by, among others the US DoD, and the EEC) demands that we do not ignore it. Many **SGML**-based systems use **TeX** (or **L<sup>A</sup>TeX**) as the document formatting engine.

But **SGML** is an existing ISO standard. An evolving standard of which we must be keenly aware is **ODA** (Office Document Architecture). One of the objectives of **ODA** is to permit the electronic interchange of documents over open systems. In the **TeX** world we would argue that this has already been achieved. One aspect of concern to **ODA** is mathematical text. Again, we would argue that this has already been achieved. Sadly, the way that national and international standards are created does not ensure that the best *de facto* standard becomes enshrined in the ultimate ISO standard. At present, the various **ODA** national committees and panels are considering the input of mathematics. The principal European submission which has been received suggests the use of **eqn**.<sup>\*</sup> This is a somewhat limited and limiting approach. It is of the utmost importance that we in the **TeX** world promote the other alternatives that we know (and love).

We can live outside the 'standards world'. It is possible. But it is uncomfortable, and ultimately it will lead to atrophy. We should ensure that decisions, like those to become part of the **ODA** standard, are made with reference to a wide spectrum of possibilities. It may be that **TeX** is inappropriate to **ODA**. But that conclusion must be reached by active and informed debate, not by ignorance and apathy.

---

\* Editor's note: A U.S. contribution has recommended examination of **eqn**, **TeX**, and **SGML**.

## An approach to the interchange of mathematical expressions

### Introduction:

There are a number of existing ways in which mathematical information may be interchanged, using only the **ASCII** character set. These include **eqn**, **TeX** (and its siblings, **L<sup>A</sup>TeX** and **A<sup>M</sup>S-TeX**), and **SGML**. There are, of course others, but they tend to be linked to some proprietary system. The three above all have the advantage of belonging, in a sense, to no-one. The first two, **eqn** and **TeX** have the further advantage of having been tested 'in the field' for a number of years. They are well understood by a large population. **eqn** is, of course, a Unix tool, distributed widely with that quasi-standard operating system. **TeX** has been implemented on a wide variety of computers (including Unix machines), and has the further advantage of belonging even more firmly to the public domain—to the extent that its algorithms often crop up in proprietary systems. Mathematical encoding in **SGML** is possible, but is found infrequently, many **SGML** systems opting out of the difficulties by adopting either **eqn** or **TeX** as their mathematical processor.

It is not the purpose of this presentation to extol the virtues of **TeX** over **eqn**. The purpose is to make evident the power of **TeX** to encode mathematical expressions in an unambiguous and straightforward way which is generally both 'human-readable' and coherent, and which may also be reasonably compact. And by implication, that any consideration of a suitable technique for the interchange of mathematical information should include examination of **TeX**.

It should be understood that 'TeX', as used here, is intended to include the common 'add-on' facilities provided by both **L<sup>A</sup>TeX** and **A<sup>M</sup>S-TeX**.

### Use of existing standards:

As noted above, **TeX** uses only the **ASCII** character set; thus it is commonly used for the transmission of technical material over existing local- and wide-area networks. This practice is some years old now. **TeX**, (unlike **eqn**), has no 'reserved words'. The sole 'reserved' character is the backslash \ (and even that is not very reserved), which is used as an 'escape character' to denote that the token (symbols) which follow should be treated in a special way. Thus **\alpha** is a way of representing  $\alpha$ . This takes us no further than **SGML**, where we could use the Public Greek Symbols Entity 'alpha'. However, it does provide us with extra tools which enable us

to write expressions like  $a \over b$  to obtain  $\frac{a}{b}$ . It is but a small step from this to something a little more grandiose like

$$A = \prod_{j=1}^{p-1} \int_0^\theta L_j = \theta^{p-2}$$

or

$$\frac{\phi(\theta_i) - \phi(\theta_{i-1})}{1 - \phi(\theta_{i-1})} = P, (i = 1, 2, \dots, n)$$

Keyboard symbols are used when appropriate (like the parentheses, the = and - signs), but all other symbols are obtained through the use of the \ operator. The T<sub>E</sub>X encoding for these two expressions is:

```
A = \prod_{j=1}^{p-1} \int_0^\theta L_j = \theta^{p-2}
```

and

```
{\phi(\theta_i) - \phi(\theta_{i-1})
 \over 1 - \phi(\theta_{i-1})}
 = P, (i=1,2,\ldots,n)
```

Apart from explaining that the ^ and \_ symbols are used to denote super- and sub- scripts, the only other point to note is the use of { and } for grouping sub-expressions.

#### Parenthetically, an apparent ambiguity:

Obtaining superscripts requires comment. T<sub>E</sub>X uses the same 'operator' for superscripting and for 'raising to a power': that is to say,  $x^2$ , where the '2' is a power is indistinguishable from  $x^2$ , where the '2' is a 'true' superscript (or superior)—they are both obtained from  $x^{\wedge}2$ . This may be seen as an ambiguity. If it is, we can point to the T<sub>E</sub>X control sequence \sp which can be used in place of ^ . Thus, although  $x \sp 2$  will give exactly the same formatted result as  $x^2$ , they would be distinguishable in the original ASCII text. In common with most computer systems, there is always a tendency to want to minimise keystrokes, so that most T<sub>E</sub>X users would tend to type ^ rather than \sp, even if what they meant was 'superscript'.

#### Extensibility:

Indirectly this also points to another feature of T<sub>E</sub>X. It allows the creation of 'macros'—combinations of more primitive commands which can be extended to provide very powerful features. As an example, we may take \matrix, a macro which allows us easily to write expressions like:

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

where the T<sub>E</sub>X code is

```
A=\left(\matrix{x-\lambda & 1 & 0 \cr
 0 & x-\lambda & 1 \cr
 0 & 0 & x-\lambda \cr
}\right)
```

For a fuller explanation here we need to add that & is the 'pseudo' tab character, which separates columns, and \cr is a control sequence which indicates the end of a row.

T<sub>E</sub>X is, in fact, a powerful programming language its own right, although this may not be an essential feature in this context. Nevertheless, it indicates that any apparently omitted features in the base language may be created for future needs, or, in fact, needs outside strict mathematical uses.

#### A de facto standard:

T<sub>E</sub>X has been used widely in the academic community (the main users of mathematical typesetting software) since 1978. It was first released in a prototype form in 1978, but underwent a substantial revision before its present form, which was completed in 1982. Since then, T<sub>E</sub>X has been ported to an astonishing variety of computers. Almost any and every machine with at least 16-bit words has a version of T<sub>E</sub>X working on it. All versions of T<sub>E</sub>X must pass the so-called 'trip' test, before they may call themselves T<sub>E</sub>X. This is intended to ensure that each and every version produces exactly the same output for the same input.

Naturally it is not the formatting qualities of T<sub>E</sub>X which are relevant here, but instead its ability to represent the full range of mathematical expressions. One indication of its success in this area is the use to which T<sub>E</sub>X has been put in teaching blind mathematicians. We know of at least two projects where this has been undertaken, with apparent success. The attractiveness of T<sub>E</sub>X here lies in its ability to linearise an expression, much as one would do if one had to 'talk' maths to a colleague, without the advantage of chalk and board.

But besides this linear quality, we must point to the widespread use and adoption of T<sub>E</sub>X, the de facto standardisation which has occurred in the academic world, and the huge pool of T<sub>E</sub>X-familiar keyboarders.

#### Conclusion:

T<sub>E</sub>X has the ability to encode mathematical expressions in a way which is both human-understandable and easily transmitted by electronic means. A great many implementations exist which can turn such encoded material into displays of one sort or another—the linear encoding is readily transformed

into a two dimensional display. Since  $\text{\TeX}$  has been available widely for a number of years, it is well understood by a large population, although, as public-domain software it lacks the outright backing of any large commercial organisation.

The approach adopted by  $\text{\TeX}$ , and the lessons which may be learned from it, can be put to good use as part of the ODA standard for mathematical encoding. It is not the contention here that only  $\text{\TeX}$  is a suitable vehicle, or even that adoption of  $\text{\TeX}$  as it exists now is the very best solution. The principal argument is that to exclude it from discussion and examination would be counter-productive and limiting.

◊ Malcolm W Clark  
Imperial College Computer Centre  
Exhibition Road  
London SW7 2BP, England  
Janet: `texline@uk.ac.ic.cc.vaxa`

---

## Teaching $\text{\TeX}$

Bart Childs

### Acknowledgements, Disclaimer, and Calls

This is a continuation of the paper on "Syllabi for  $\text{\TeX}$  and METAFONT courses", pages 117-127 in the Proceedings of the 9th TUG meeting. This paper contains some of the details of the previous paper in tabular form. The errors are mine, but many of the ideas are those of the teams who did the work. I would especially like to thank Barbara Beeton, Dean Guenther, Pierre MacKay, and David Ness for their continual reading, editing, and other input. Don Knuth also reviewed an earlier draft and said he is no longer a "10". Well ...

This paper includes a flow chart to illustrate the prerequisites of  $\text{\TeX}$ ,  $\text{\LaTeX}$ , and METAFONT classes. A two page table is used to show the contents of the  $\text{\TeX}$  courses, and another table shows the contents of the  $\text{\LaTeX}$  courses. The  $\text{\LaTeX}$  overview was not written by  $\text{\LaTeX}$ perts. The three level test for  $\text{\TeX}$  has been revised several times.

I am **calling** for input of the following form:

1. constructive criticism of this document, especially the  $\text{\LaTeX}$  contents;
2. submission of additional questions for the  $\text{\TeX}$  tests;

3. submission of questions for the (yet to be done)  $\text{\LaTeX}$  tests; and
4. your suggestions for any other items that you think will help.

I will make these sources available for all. I intend to incorporate input and reissue this document on a periodic basis. The rest of this paper is what I would distribute.

### Overview

A user should be familiar with the use of a text editor before undertaking the *Beginning  $\text{\TeX}$*  course. In some cases the user can employ a word processor and store the file as an ordinary text file if such an editor is the user's common means of creating a file.

The *Beginning  $\text{\TeX}$*  course should give the student an understanding of the basic nature of  $\text{\TeX}$  and the parameters it uses in producing attractive documents. After the course, the student will feel comfortable taking examples from *The  $\text{\TeX}$ book* for use, but may not yet be fully at ease modifying these examples.

Upon completion of the *Intermediate  $\text{\TeX}$*  course, the student should be able to adapt and modify examples from *The  $\text{\TeX}$ book* to suit individual purposes. He or she will also be able to develop creative solutions to typesetting problems using  $\text{\TeX}$ .

The *Advanced  $\text{\TeX}$*  course should give the student knowledge of how many of the examples in *The  $\text{\TeX}$ book* are created. Further, the student should be able to create new macros and documents using these concepts. At this stage of knowledge,  $\text{\TeX}$ 's capability as a 'text-oriented programming language' can be exploited.

### Beginning $\text{\TeX}$

This course provides a practical introduction for those with limited, or no, exposure to  $\text{\TeX}$  and will be composed of about equal parts lecture and "hands-on" sessions, including many practical exercises for each object of study. Participants will be introduced to  $\text{\TeX}$  as a language for typesetting, also learning its context in the history and milieu of word-processing and typesetting.  $\text{\TeX}$  is compared with other popular formatting systems such as word-processors and desktop publishing systems.

$\text{\TeX}$  concepts to be covered include: methods of preparing simple paragraphs, changing line spacing and specifying fonts; simple boxes, characters and accents; justification and line breaking. In math mode, superscripts, subscripts, and fractions will be addressed.

Each registrant will be given copies of *The T<sub>E</sub>Xbook* and *First Grade T<sub>E</sub>X*.

Prerequisite: familiarity with a text editor is essential.

### Intermediate T<sub>E</sub>X

This course comprises equal parts lecture and laboratory sessions, including many practical exercises. It builds upon the foundation laid at the beginning level.

Topics to be covered include: more complicated paragraph shapes, paragraphs with labels, hanging indentation; more complex interaction between glues and boxes; greek letters, special symbols and delimiters in math mode; displayed equations; control of line and page breaks; simple tables.

Prerequisite: Beginning T<sub>E</sub>X or equivalent knowledge.

The student will furnish his/her own copy of *The T<sub>E</sub>Xbook*.

### Intensive T<sub>E</sub>X

This course is a combination of the above two courses. It is taught at a high speed in approximately one week.

### Advanced T<sub>E</sub>X

This course is designed for *all* experienced T<sub>E</sub>X users and includes both lectures and experimentation. This course will give an intensive study of macro writing and designing macro packages.

Topics will include: detailed explanation of the relationship of boxes (`\vbox`, `\vtop` and `\hbox`) and glue; usage of registers, especially box registers and counter registers; basic concepts and ideas of macros; use of `\halign` in constructing tables and equation arrays in math mode; loading fonts, magnification, kerning, ligatures; controlling the line and page breaking algorithms; delimited and undelimited macro parameters; global *vs.* local definitions; conditionals, loops, and counters; tools such as `\let`, `\futurelet`, `\chardef`, `\catcode`, and `\begingroup`; expansion of tokens, and when such expansion takes place. We will design macros in class and analyze common constructions, with practice in interpreting existing macros so that they may be customized for special applications.

Prerequisite: Intermediate T<sub>E</sub>X or equivalent knowledge.

### Course Contents

Following the tests below are tables showing several topics concerned with T<sub>E</sub>X and typesetting. The tables attempt to indicate the suggested detail in

which these topics are covered in each of the three courses.

### The T<sub>E</sub>X Test

The test is divided into three levels. The first level is intended for students who have completed the **Beginning T<sub>E</sub>X** course. Upon completion of the course, you should be able to answer at least 75% of the questions with correct answers. We would hope that it could be done without reference to the *T<sub>E</sub>Xbook*, *First Grade T<sub>E</sub>X*, or other sources. With these sources and an extra thirty minutes or so you should be able to answer all the questions, regardless of your instructor.

Performance at this level on the first test should be a prerequisite for taking the **Intermediate T<sub>E</sub>X** course. Similarly, an equivalent level of performance on the second test should be a prerequisite for the **Advanced T<sub>E</sub>X** course. Completion of the third course should lead to a good score on the third level of the test.

We realize that a lot of T<sub>E</sub>Xers have been self taught. We feel the tests could be used as effective self-tests after independent study of the *The T<sub>E</sub>Xbook* or another such manual.

Many of the questions contain fragments of T<sub>E</sub>X code. These fragments are in the typewriter font and their lines are numbered. Ellipses (...) are used liberally to indicate that more T<sub>E</sub>X material may be present.

These tests and other teaching materials are the property of the T<sub>E</sub>X Users Group. They may be used freely for the purpose of expanding knowledge about T<sub>E</sub>X systems as long as proper credit to their source and acknowledgment of the goals and purpose of the T<sub>E</sub>X Users Group are prominently displayed.

The T<sub>E</sub>X Users Group solicits contributions and opinions on these materials. We intend to reissue these materials on a regular basis with updates containing new contributions.

The tests are based on levels of expertise varying from 0 to 10. Level 0 corresponds to knowing nothing about T<sub>E</sub>X while Donald Knuth is level 10.

### The T<sub>E</sub>Xtest — Level One

1. One of the visible ASCII characters is used as T<sub>E</sub>X's escape character. It is the `___` symbol and its name is \_\_\_\_\_.

2. In the following T<sub>E</sub>X code fragment:

1 ...

- 2 last line of a paragraph.  
 3 `\parskip=6pt`  
 4 First line of a new paragraph  
 what horizontal and vertical spaces will be between "...a paragraph." and "First line...?"
3. What TeX control sequence is the equivalent of a blank line?
  4. How do you cause the TeX program to execute and process the file "testfile.tex" on your system?
  5. When TeX has finished processing "testfile.tex", how can you get another look at the error messages (with more detail) without running TeX again?
  6. The code fragment "the TeX program" produces output "the TeXprogram" which is obviously missing a space after the TeX logo. Give two or more ways to correct this.
  7. What is the name of TeX's monospaced font and what control sequence is used to access it?
  8. What is the typographer's name for straight lines?
  9. How do you end the indentation from the `\narrower` instruction?
  10. How should you end the current paragraph before ending the `\narrower` mode?
  11. What is the indentation of the following paragraph and why?
 

```

1  {\narrower\narrower
2  first line of a paragraph.
3  ...
4  last line of a paragraph.
5  }\par
```
  12. Consider the following code fragment:
 

```

1  \parindent=0.5in
2
3  A first paragraph ...
4  \parindent1.0in
5
6  A second paragraph ...
7  \bye
```

How much will each of the paragraphs be indented? first \_\_\_\_\_ second \_\_\_\_\_
  13. How do you specify an italic correction?
  14. What does an italic correction do?
  15. Is the space in:
 

```

1
2  \centerline {Centered}
3
```

necessary \_\_\_\_\_, optional \_\_\_\_\_, or in error \_\_\_\_\_?

16. What will TeX output from the following code fragment?
 

```

1
2  \centerline Center This!
3
```
17. Describe the output of this code fragment?
 

```

1  \bf{this is bold text} ...
```
18. What is a widow?
19. How do you place the page number flush right in a running head?
20. How do you keep the left margin fixed and move the right margin to the left by 0.5in?

A new TeX user has decided to create some macros. The following definitions are OK or BAD! Mark each of these OK or BAD, and indicate what is wrong with the BAD ones. Assume the plainest of TeXs.

21. `\def\A1{...}`
22. `\def\A-OK{...}`
23. `\def\Test{...}`
24. `\def\{...}`
25. `\def\10{...}`

There are ten visible ASCII characters that TeX has reserved for special uses. For example, the dollar sign is used to toggle mathematics mode. List the other nine and their use as illustrated.

26. \$ toggle math mode
27. \_\_\_\_\_
28. \_\_\_\_\_
29. \_\_\_\_\_
30. \_\_\_\_\_
31. \_\_\_\_\_
32. \_\_\_\_\_
33. \_\_\_\_\_
34. \_\_\_\_\_
35. \_\_\_\_\_

### The TeXtest — Level Two

1. What *mode* is TeX in when building a paragraph?
2. How do you end a `\topinsert`?
3. If TeX hyphenates a word badly, how do you fix it?
4. What happens to a paragraph that has both normal indentation and a `\hangindent` specification?
5. What will `\line{A Short Line}` look like in a normal page?



6. Why won't a field like `{\hfil x \hfil}` be centered in a `\settabs` environment?
7. How can you get a black square, like , in the middle of a line of text?
8. Consider the code fragment:
 

```
1  ...
2  \eject\vskip2in
3  How now brown cow...
```

 Where is the "How now brown cow" placed relative to the top margin of the page?
9. How do you move a `\vbox` to the right one inch?
10. How do you reduce or prevent widows?
11. What is a penalty?
12. What happens when you forget to end a `\footnote`?
13. What happens when you try to end a document without a proper end to an insert?
14. What happens if you have a blank line in display math mode?
15. How is the `\tabskip` parameter used?
16. How are the `\lineskip` and `\lineskiplimit` parameters used?
17. What actions should you consider to correct the conditions that caused the warning `Overfull hbox?`
18. What actions should you consider to correct the conditions that caused the warning:
 

```
1  Underfull \vbox has occurred
2  while \output is active
```
19. Show how to assign the control sequence `\8` to a new font "cmss8" that has not already been defined in plain `TeX`.
20. What is the meaning of `#1` in a macro definition?
21. Arguments to macros may be *delimited* or *undelimited*. Describe how `TeX` determines arguments in the two cases. How do users notice the difference?
5. Are the leading spaces in `\halign` entries significant?
6. Are the trailing spaces in `\halign` entries significant?
7. Are the leading and trailing spaces in `\settabs` entries significant?
8. How serious will underfull `vbox` badness 3412 be?
9. Build a macro called `\xx` that has one parameter delimited by a semicolon. The macro is to center its one parameter and set it in bold.
10. What happens when `\def\p#1{\it\centerline#1}` is called with `\p01234`?
11. The following code fragment is an exercise in being careful:
 

```
1  \newcount\cntr
2  \advance\cntr1\the\cntr1
```

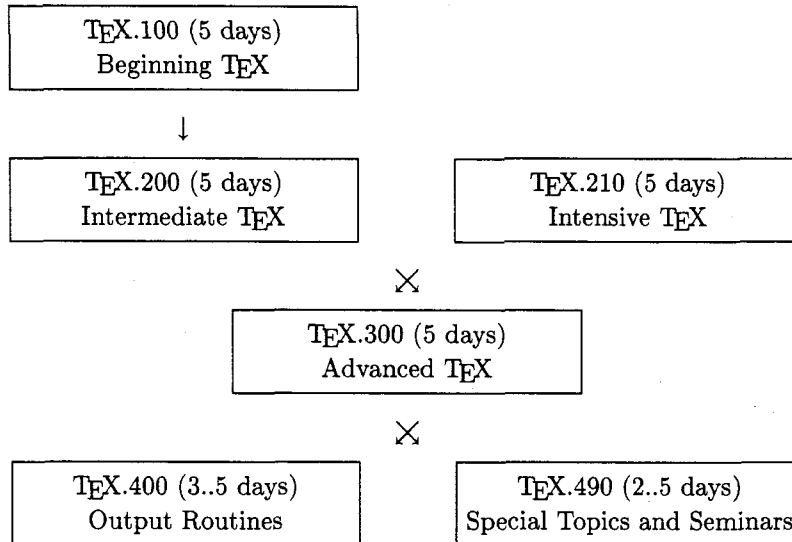
 What is the resulting value of the counter?
12. How can you zero all the dimensions of `box0`?
13. What information is in a `tfm` file?
14. What information is stored in a `pk` or `gf` file?
15. How are `\vtop` and `\vbox` similar? How do they differ?
16. A two column macro package works by gathering enough information for both columns before invoking the `\output` routine. What is the name of the `TeX` primitive that is probably used to determine each column?
17. What does `\futurelet` allow you to do?
18. What `TeX` commands would you have to use to automatically build an index and/or table of contents to a separate file, and print it in the output?
19. What happens when you underestimate the number of lines in a `\parshape` command?
20. Under what conditions can you use the built-in fonts of an arbitrary printer?
21. What characteristics should you look for on a page of output to try to determine if the page was prepared using *PageMaker*, *troff*, ..., or `TeX`.
22. What element(s) of `TeX` is (are) case insensitive?

### The `TeX`test — Level Three

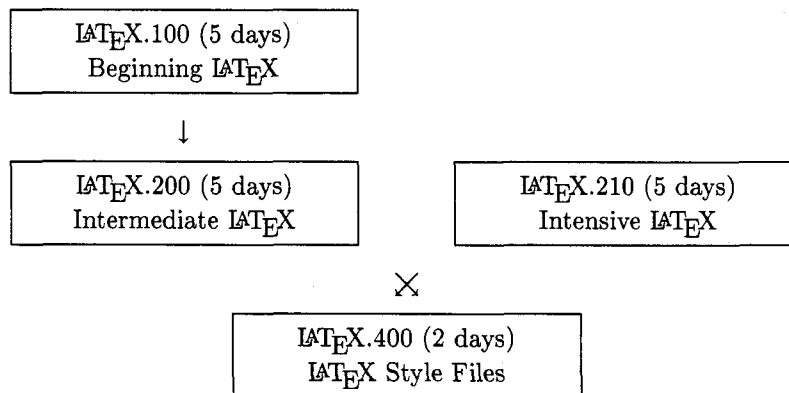
1. How is `TeX`'s escape character determined?
2. What is the name of the control sequence that can be used to accomplish the function of the left brace, `{`, inside a macro that will not have its matching right brace, `}`?
3. What are the names of the parameters that specify the amount of glue above and below display math?
4. `TeX` treats several consecutive spaces as one. Thus the usual practice of keyboarding 2

## TITLES and PREREQUISITE STRUCTURE

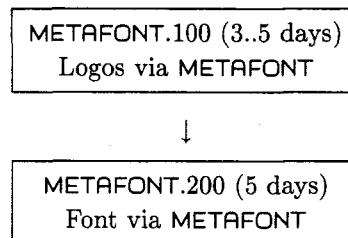
### $\TeX$ Courses



### $\LaTeX$ Courses



### METAFONT Courses



T<sub>E</sub>X Courses — Contents

Beginning	Intermediate	Advanced
<b>Typesetting</b>		
Typesetting Milieu, Design and Typesetting Dimensions, T <sub>E</sub> X and WYSIWYG	What you should unlearn (Underlines, ...), Magnification	
<b>Design</b>		
margins, typesize, \leftskip, \rightskip, \narrower, \parindent	penalties and affecting design, \looseness, \tolerance	Database driven design, interface between T <sub>E</sub> X and other worlds, \pagegoal, \prevgraf
<b>Programming</b>		
Public domain, why pay, WEB Use of ASCII keyboard How it runs	It is a programming language	WEB and internal structure
<b>T<sub>E</sub>X and Other Things</b>		
T <sub>E</sub> X vs. L <sup>A</sup> T <sub>E</sub> X vs. Script vs. vs. Pagemaker vs. WORD ...	What can and cannot be imported and exported	
<b>Markup Language</b>		
They exist, What is plain Primitives vs. Macros	plain.tex as info source, A <sub>M</sub> S-T <sub>E</sub> X, L <sup>A</sup> T <sub>E</sub> X what - why?	Designing your own
<b>Syntax</b>		
{}, do they surround or follow	Spaces that behave unexpectedly	Why \obeylines works like it does, \obeyspaces, verbatims
<b>Spacing</b>		
Significant / Insignificant Spaces Tilde, Slash, Space \hskip, \vskip, \baselineskip	\vglue, \kern, \hbox, \vbox, \vspace, \vglue, \hspace, \thinspace	letterspacing, sidebaring
<b>Glue</b>		
Dimensions, Terminology	Stretchability / Shrinkability Negative glues	Output Routines \splittopskip, \splitmagstep, \vsplit, \vadjust, \unhbox, \unvbox
<b>Boxes</b>		
Primarily in Error Messages Boxes have height, depth, and width	moving boxes around \raise, \lower, \moveright, ... What \hbox \vbox are Stephan's \hboxr \vboxr	Understanding Stephan's boxing
<b>Rules</b>		
\hrule	\hrules and \vrules <i>au naturel</i> \?rules for \struts \?rules for boxes	
<b>Fonts</b>		
What is a font, CM family What does T <sub>E</sub> X needs to know Other fonts ?	What are sources of fonts Scaling and metrics pk vs. p <sub>x</sub> l vs. gf Proprietary fonts, their limits	Introduce METAFONT TFxxxxx PKxxxxx etc.
<b>Paragraphs</b>		
Paragraphs, especially \parindent, \parskip, \par	\narrower, \hangindents, \items	\parshape, \prevgraf
<b>Lines</b>		
\centerline, \leftline ... \line	line / paragraph interactions + meaning	everyline?
<b>Math</b>		
Display math as paragraph suspender In-line math	Subscript Superscript (incl use as footnote numbers) \eqalign and other math stuff	Broken equations Special math spacing Special math fonts

T<sub>E</sub>X Courses — Contents

Beginning	Intermediate	Advanced
<b>Macros</b>		
Macros as shorthand	Macro with parameters, delimiters	<code>\unskip</code> , <code>\outer</code> , <code>\xdef</code> , <code>\gdef</code> Combinations of macros Macro structure and exceptions
<b>Debugging</b>		
Simple debugging Putting in artificial ends ...	Purposeful errors <code>\showthe</code> , <code>\showbox</code> , <code>\show</code>	<code>\tracing...</code> and <code>\showbox</code> Visible boxes, <code>\tracingall</code>
<b>Errors</b>		
Flesh wounds, Fatal errors, Misunderstandings Which can be ignored ?	When errors can be understood	Genuine obscurities
<b>Tabs and Alignment</b>		
<code>\settabs</code> , <code>\tabalign</code> , <code>\cr</code> Copy alignment from T <sub>E</sub> Xbook and Use it	Copy an alignment and modify it <code>\hidewidth</code> , <code>\omit</code> , <code>\strut</code> , <code>rules</code>	Create an alignment Alignments and Rules
<b>Penalties</b>		
Notice that they exist <code>\hyphenpenalty</code>	Penalties in formatting <code>\goodbreak</code> ...	All penalties — What they really do
<b>Output Routines</b>		
No mention except for <code>\hoffset</code> , <code>\voffset</code> , <code>\footnotes</code>	<code>\footnotes</code> with numbers	[We had nothing?]
<b>I/O Management and Files</b>		
Comments, Documentation etc.	<code>\input %</code> to get rid of spaces	writes index table of contents
<b>Modes</b>		
No mention other than <code>\\$</code> confusion	<code>\hmode</code> vs. <code>\vmode</code> what and how Math modes, Restricted modes	<code>\ifvmode</code> ...
<b>Inserts</b>		
<code>\topinsert</code>	<code>\midinsert</code> , <code>\pageinsert</code> , insert interaction	
<b>Chars</b>		
No Mention	<code>\def\xx{\char..}</code>	Redefine Chars
<b>Graphics and T<sub>E</sub>X</b>		
	Space for graphics, <code>\boxit</code>	PiCT <sub>E</sub> X other things available, Manual, L <sup>A</sup> T <sub>E</sub> X Circle and Line Fonts, Rounded Boxes
<b>Tokens</b>		
No mention	No mention	Explain tokens
<b>Font Families</b>		
No mention	Mention	Understand and create
<b>Control Structures</b>		
Grouping simple existing <code>\ifs</code>	<code>\begingroup</code> , <code>\endgroup</code> , <code>\ifs</code> modifying existing <code>\ifs</code> , create <code>\newcount</code> , <code>\newdimen</code>	<code>\new...</code> , <code>\bgroup</code> , <code>\egroup</code> , <code>\repeat</code> , creation of <code>\ifs</code> <code>\futurelet</code> , <code>\expandafter</code> , <code>\afterassignment</code> , <code>\noexpand</code>
<b>PotPourri—Anomalies, Etc.</b>		
		T <sub>E</sub> X and SGML dvi and PostScript Graphics, availability, ...

L<sup>A</sup>T<sub>E</sub>X Courses — Contents

Beginning	Intermediate
<b>Typesetting</b>	
L <sup>A</sup> T <sub>E</sub> X, Typesetting milieu, Design, Dimensions	What you should unlearn (underlines, ...)
<b>Design</b>	
margins, typesize, \leftskip, \rightskip, \narrower, \parindent	Penalties affecting design, \looseness, \tolerance
<b>L<sup>A</sup>T<sub>E</sub>X and Other Things</b>	
L <sup>A</sup> T <sub>E</sub> X vs. Script vs. L <sup>A</sup> T <sub>E</sub> X vs. Pagemaker vs. WORD ... Public domain, written in T <sub>E</sub> X, Use of ASCII keyboard, How it runs	What can and cannot be imported and exported A <sub>M</sub> S-T <sub>E</sub> X, L <sup>A</sup> T <sub>E</sub> X - what do they do, and compare to a markup language
<b>Syntax</b>	
{}, []'s, and \begin - \end	Spaces that behave unexpectedly
<b>Spacing</b>	
Significant/Insignificant spaces, Tilde, Slash, Space, \hspace, \vspace, \hfil	*'d as opposed to not
<b>Glue</b>	
Dimensions, Terminology	Stretchability/Shrinkability, Negative glues
<b>Boxes</b>	
Only in error messages	\mbox, \makebox, \fbox, \framebox
<b>Rules</b>	
\rule	
<b>Fonts</b>	
What is a font, CM family, What does L <sup>A</sup> T <sub>E</sub> X need to know, Governed by logical structure	\newfont is rare, math fonts are different
<b>Paragraph Environments</b>	
Quotations, centering, verbatim, verse	\narrower, \hangindents, \items
<b>Line Environments</b>	
flushright, flushleft, \raggedright	Line/paragraph interactions & meaning
<b>List Environments</b>	
Itemize, enumerate, description	[More]
<b>Math Environments</b>	
In-line and display	[Much more]
<b>Environments in general</b>	
	\picture and more on the others
<b>Styles</b>	
Discuss article, book, letter, report, etc.	
<b>Definitions</b>	
Commands (macros) as shorthand	\newcommand, \newenvironment, \renewcommand
<b>Debugging</b>	
Simple debugging, putting in artificial ends ...	Purposeful errors, \showthe, \showbox, \show
<b>Errors</b>	
Flesh wounds, Fatal errors, Misunderstandings, When not to worry about content	Special L <sup>A</sup> T <sub>E</sub> X errors
<b>Tables</b>	
Arrays and tabular alignment	
<b>Penalties</b>	
Notice that they exist, Errors fall through to T <sub>E</sub> X	Penalties in formatting
<b>I/O Management and Files</b>	
Comments, Documentation, etc.	\includes
<b>Inserts</b>	
Some objects (tables and figures) float	
<b>PotPourri—Anomalies, Etc.</b>	
What is L <sup>A</sup> T <sub>E</sub> X's meaning of objects?	

## HI-TeX Cutting & Pasting

Michael Ballantyne  
Michael Spivak  
Yoke Lee

### The Problem

As more and more macro packages are written for TeX, the problem of exceeding TeX's memory capacity becomes more common and more acute. In fact, versions of TeX with larger memory have already been created to help alleviate this problem, but such versions don't run on the personal computers that most of us have.

Some macro packages try to skirt the problem by selectively loading only needed subsets, but this strategy can fail when many different sorts of constructions are required in the same file. It remains true, however, that different collections of macros are normally required for different parts of a file, with insuperable problems arising only when several different collections need to be loaded at once.

For example, a large macro package might succeed in typesetting individual complicated tables, but cause TeX's memory limits to be exceeded when used in conjunction with other macro packages, or when numerous tables have to be held over for inserts. In this case, as a last resort, one could: (1) make a special file to individually typeset all the tables required for a book or paper, one to a page; (2) leave the proper amount of blank space in the main file for each table; (3) print the two files, cut the tables from the special file, and paste them into the blank spaces in the main file.

Though an analogous procedure is required when a photograph has to be inserted in a book, it seems singularly unattractive when the inserted material is just more text that has already been typeset by TeX. But some of the allure may be restored when the computer is used to do the cutting and pasting.

### The DVIPASTE Solution

Our "solution" involves a little macro package, `dvipaste.tex`, and a C program, `dvipaste.c`. In the case of the table example discussed above, we would first make a file, say `tables.tex`, of the form

```
\input dvipaste
\input {macros for tables}
\setbox0\hbox{(table 1)}
\sendout{\box0}
\setbox0\hbox{(table 2)}
```

```
\sendout{\box0}
...
\end
```

When this file is run through TeX, it will produce `tables.dvi`, and an auxiliary file `tables.dat`. Printing `tables.dvi` will produce the various tables, one to a page, each positioned at the bottom left corner of the page. The file `tables.dat` will contain a sequence of lines like

```
123.45pt .4pt 233.567pt
```

where line  $n$  contains the height, depth, and width of the table on page  $n$ .

Now the main file, say `book.tex`, will also have `\input dvipaste` at the beginning, but here each table will be replaced by

```
\paste{tables}{n}
```

where  $n$  is the number of the page on which the desired table is printed in the `tables` file. A `\paste{...}{...}` can appear anywhere, for example, as

```
\centerline{\paste{tables}{n}}
```

or

```
\midinsert{\paste{tables}{n}}
```

etc. TeX will replace each such `\paste` command with a blank box having exactly the right height, depth and width (which it reads from `tables.dat`), at the same time inserting an informative little `\special`, which most drivers will happily ignore. When `book.dvi` is printed, exactly the right amount of space appears for each table.

It would appear that we haven't done much more than the procedure outlined in the previous section, except that the amount of blank space for each table has been measured for us by TeX. Now, however, we can use the `dvipaste` program,

```
dvipaste book newbook
```

to use the file `book.dvi` to produce a new file `newbook.dvi`. In creating this new `.dvi` file, `dvipaste` will examine the `tables.dvi` file and extract `.dvi` commands to be placed at the position of the various `\special`'s that were inserted by the `\paste`'s. These extra `.dvi` commands have the effect of causing the tables to be printed in precisely the places occupied by blank spaces in `book.dvi`. Thus, `newbook.dvi` will print exactly what `book.dvi` would have printed if the table specifications had been part of `book.tex` (which is not to say that `newbook.dvi` is exactly the same file that `book.dvi` would be).

Although this solution may not be ideal, it involves only an insignificant amount of extra time,

not to say that `newbook.dvi` is exactly the same file that `book.dvi` would be).

Although this solution may not be ideal, it involves only an insignificant amount of extra time, since `dvipaste` runs much quicker than `TEX`, and little extra work. It's true that an extra file is required, but this isn't an overwhelming inconvenience—it might even be *more* convenient to keep all the tables in a separate file. This illustration used a single auxiliary file `tables.tex`, but any number could actually be used.

### How it Works

`\sendout#1`, defined in `dvipaste.tex`, writes a line to the `.dat` file giving the height, depth and width of `#1`, and then adds a `\vskip` down to the bottom of the page, followed by

```
\special{beginpaste:}%
\noindent\rlap{\smash{#1}}%
\special{endpaste:}%
\vrule height1sp width1sp depth0pt
\ejct
```

The `\vrule` is obviously not meant to be seen. The only important thing is that it's *there*, more precisely that it's *here*, right back at the point where the `\noindent` began. This means that the `.dvi` commands between the two `\special`'s will create the table seen on the page, *starting and ending* at the lower left corner of the table.

On the other hand, `\paste{(subfile)}{n}` expands to

```
\special{dvipaste: (subfile)n}\vbox...
```

where the `\vbox...` is a blank box with the height, depth and width given on line `n` of `(subfile).dat`. (The first `\paste` with the argument `(subfile)` causes `TEX` to open the file, and store all the information in an appropriate place; subsequent uses merely ferret out that information.) The `dvipaste` program looks for such specials, and replaces them by the relevant `.dvi` commands on page `n` of `(subfile).dvi`. Of course, it's a bit more complicated than that, because each font declaration in a `.dvi` file must be made just once (before the `postamble`), so font declarations from a `(subfile).dvi` must be deleted if they have already appeared in the main file, renumbered if they declare new fonts, etc.

### Extensions

Although current drivers will presumably ignore a `\special{dvipaste:}` command, they don't *have* to! In fact, screen and printer drivers could perform the same maneuvers as `dvipaste`. A screen driver of this sort would preview the complete book, with the

tables inserted, without using `dvipaste`. `dvipaste` itself might be reserved for the final run, before the `.dvi` file is sent off to the typesetter.

### Availability

`dvipaste.c` and `dvipaste.tex` are copyrighted in the GNU spirit (they are distributed for a nominal charge, and must be passed on according to the same terms). For a standard IBM 360K double-sided diskette containing `dvipaste.tex`, `dvipaste.c` and an MS-DOS executable `dvipaste.exe`, send \$4.00 to `TEXplorators`, 3701 W. Alabama, Suite 450-273, Houston, TX 77027.



## Another Dingbat Idea

I take pen in hand to describe the design and coding of a simple dingbat. I hope that this will inspire all you would-be METAFONTers to try your hands, heads and keyboards at creating entries for the Dingbat Competition, announced in the last issue of *TUGboat*.

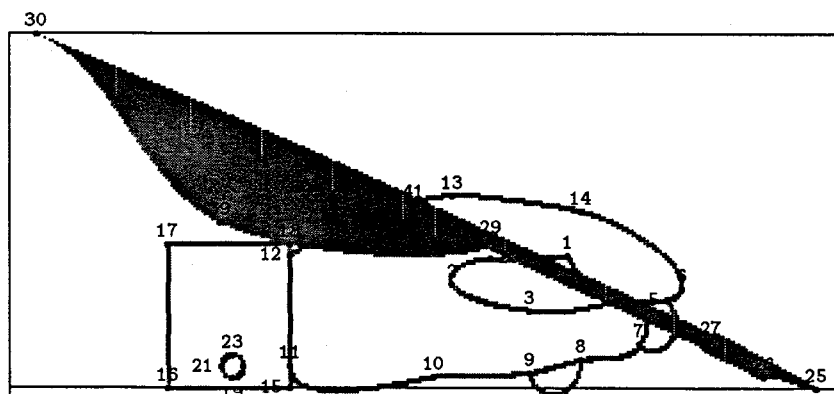
I started with a sketch of a left hand holding a quill pen (left, because I had to draw with my right) and marked what I figured would be the necessary points to describe the figure with METAFONT. Of necessity, the figure is simplistic, somewhat cartoonish; remember that all the details that you lovingly describe on your initial large sketch (mine was approximately 180 points high and 480 points wide) must survive down to 10 points!

The next step was to lay out a grid and orient the figure on it relative to the baseline and width. Since I do not want any of the character to lie under the line of text next to which it occurs, I oriented the bottom of the hand and the cuff exactly on the baseline. However, I did add a wee bit of descender for the reverse video case. And there's no need for

any tricky machinations in terms of part of the figure lying to the left of the line  $x=0$  or to the right of  $x=\text{charwidth}$ ; the apparent width of the character is the real width we want  $\text{T}_\text{E}\text{X}$  to know about.

My next step was to divide the total width and height into some reasonable number of units. One caveat here: don't make the grid too fine, or you'll tend to overdo the number of points you select as key points and over-analyze the character. Think of the design process as a collaborative effort with METAFONT, rather than as an attempt to control it. As Don Knuth has said, some of the most fruitful parts of design can occur when you let METAFONT "have its own head".

Now, I selected my key points. I cannot stress too much the need to be a little freewheeling from now on. The fact that I believed a point was key at this early stage of the design process ought not to force me to keep that point in later on. In a bit, you'll see the code for the figure, where I have left the numbering of the points as I originally did them to illustrate this. You'll notice some gaps between  $z30$  and  $z41$ .



$z22 = z19 + (4.3, 4.3);$   
 $z28 = z4 + (0, 0);$

FIGURE 1: Proofmode drawing of character.



My original sketch involved a much fancier feather on the quill, which just didn't work at a design size of 10 points. I opted for a plainer feather, and removed a number of points. Figure 1 shows a proof mode of the characters as finally produced, with the numbering of the points as shown in the code.

I was now ready to start writing stuff that resembles METAFONT code; and this is it:

```
% define points for hand
x1=10.75/16w; y1=3/8h;
x2=8.5/16w; y2=2.35/8h;
x3=x9=10/16w;
y3=1.75/8h; y9=0.35/8h;
x4=11.75/16w; y4=2/8h;
x5=12.65/16w; y5=2/8h;
x6=12.95/16w; y6=2.25/8h;
x7=12.125/16w; y7=1/8h;
x8=11/16w; y8=0.65/8h;
x10=8.125/16w; y10=0.25/8h;
x20=9.25/16w; y20=2.95/8h;
x13=8.5/16w; y13=4.35/8h;
x14=11/16w; y14=4/8h;
% two points on the wrist that touch
% the cuff, and the cuff
x11=x12=x15=x18=5.35/16w;
y11=0.5/8h; y12=3/8h;
y15=0; y18=3.25/8h;
x16=x17=3/16w; y16=y15; y17=y18;
% define the button
x19=x23=good.x 4.25/16w;
y19=good.y 0.25/8h;
y23=y19+0.5/8h;
x19=1/2[x21,x22];
x21=x22-(y23-y19);
y21=y22=1/2[y19,y23];
x25=15.5/16w; y25=0;
x26=14.5/16w; y26=0.25/8h;
penpos27(quillWidth,50);
penpos28(quillWidth,50);
penpos29(quillWidth,50);
penpos41(quillWidth,50);
z28=z4; x29=x20;
x27=13.5/16w;
z29=whatever[z25,z28];
z27=whatever[z25,z28];
x30=0.5/16w; y30=h;
x39=4/16w; y39=3.75/8h;
x41=7.75/16w; z41=whatever[z29,z27];
```

You'll guess that my grid was 8 units high and 16 units wide. You'll note, too, that the leftmost point

is just a bit greater than 0, and the right a bit less than  $w$ ; this will account for sidebearings at either side of the character. Most all these points are stated in terms of the grid, rather than in terms of relation to one another; but remember to use such relationships whenever they are pertinent to the design. For example, the last two lines above define where  $x41$  and  $y41$  lie; but what is important is not the precise location of  $y41$  on the grid, but the fact that the point lies somewhere on the line between  $z29$  and  $z27$ . Needless to say, don't be shy about articulating the precise nature of these relationships in comments in your code.

Once I had established the location of all the key points on the character, I was reminded of one of my favorite Monty Python sketches: a Shakespearean actor elucidates on his craft thus: "It's not just a question of the number of words. You have to get them all in the right order." This is pretty much the next step in our design process: I have established a reasonable number of points, and now have to get them all in the right order, by writing the code to connect them in pleasing ways. This is the code I came up with:

```
% draw the hand
pickup pencircle scaled penWidth;
draw z1---z20{left}..z2..tension1.6..z3..z4
&z4{left}..tension 1.6..z1;
draw z12..tension 1.6..z13..
tension 1.8..z14..
tension 1.6..z6..z5..{left}z4;
draw z4{right}..z7..z8..tension 1.3..z9..
tension 1.4..z10..tension 1.4..{up}z11;
draw z5{(1,-1)}..{(-1,1)}z7;
draw z8{down}..{up}z9;
% draw the cuff
draw z18--z17--z16--z15--cycle;
% draw the button
draw z19..z21..z23..z22..cycle;
% draw the quill
filldraw z27r--z25
&z25{z28r-z27r}
..tension 2..{(-1,-1)}z26
&z26..{z28l-z27l}z27l
&z27l--cycle;
penstroke z29e--z27e;
fill z29r---z41r..z30
&z30{z29-z30}..z39..tension1.4..z29l
&z29l--cycle;
```

Niceties like amount of tension between points or direction desired entering or leaving points are (at

least in my experience) only rarely coded correctly first time out. This is where you and METAFONT get to work closely together. Draft some code, see what METAFONT does with it, and then tune on the basis of what you see. Often, you'll be pleasantly surprised with improvements that sneak into the design as you work.

But, of course, the code above is not yet ready for a collaborative effort with METAFONT. We have to attend to some housekeeping first. If you start the lines above with

```
beginchar("A",w#,cap#,desc#);
```

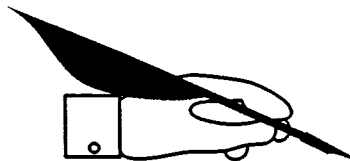
and conclude them with `endchar`; and preface them with some code that specifies font size, width, pens and so on:

```
%% Pen and character box info.
%% Set mode for device to print on
\mode=localfont;
mode_setup;
font_size 60pt#;
em#:=60pt#; cap#:=.95em#;
desc#:=.025em#; w#:=16/7em#;
overshoot#:=.025em#;
penWidth#:=em#/60; quillWidth#:=em#/20;
define_pixels(em,cap,desc,w,overshoot);
define_blacker_pixels(penWidth);
define_blacker_pixels(quillWidth);
```

and since you'll probably at some point want to see a proofmode character printed out with all the points numbered, include the lines:

```
labels(1,2,3,4,5,6,7,8,9,10,11,12,13,
       14,15,16,17,18,19,20,21,22,23,
       24,25,26,27,28,29,30,39,41);
```

However, with the mode set above, you won't get a proofmode character, but a character suitable for printing on your local device, namely:



So far, so good. But, I knew I wanted a right hand version as well (in fact, *all* I really wanted was the right hand version!) I did not even briefly consider recalculating the positions of all the points to flip the character. I could have simply copied all the code for the character above, given it a new code number and concluded it with a `rotatedabout`; but it seemed

much tidier to make the code for the dingbat proper a macro. So, start the lines of code describing the dingbat not with `beginchar` but with

```
def HandWithQuill=
```

and conclude them, not with an `endchar` but with

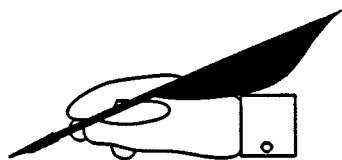
```
enddef
```

The code for the character with code "A" shown above is then condensed to

```
beginchar("A",w#,cap#,desc#);
HandWithQuill;
endchar;
```

and a mirror image version (the sought-after right hand dingbat) is coded as

```
beginchar("B",w#,cap#,desc#);
HandWithQuill;
currentpicture:=currentpicture
  reflectedabout ((0,0),(0,h))
  shifted (w,0);
endchar;
```



Since it is so easy to manipulate the `currentpicture`, we might as well produce a reverse video of the original left hand dingbat, with this code:

```
beginchar("C",w#,cap#,desc#);
HandWithQuill;
cullit;
picture savecurrent;
  savecurrent=currentpicture;
clearit;
% blacken the whole character box
fill (0,-desc)--(w,-desc)--
  (w,cap+overshoot)--(0,cap+overshoot)
  --cycle;
currentpicture:=
  currentpicture-savecurrent;
endchar;
```

(Note well the `cullit` and `clearit`!) and produce




and code a right-handed version like so:

```
beginchar("D",w#,cap#,desc#);
HandWithQuill;
% Flip the image first
currentpicture:=currentpicture
  reflectedabout ((0,0),(0,h))
  shifted (w,0);
cullit;
% Save the flipped image
picture savecurrent;
  savecurrent=currentpicture;
clearit;
% Blacken the character box
fill (0,-desc)--(w,-desc)--
  (w,cap+overshoot)--(0,cap+overshoot)
  --cycle;
currentpicture:=currentpicture-savecurrent;
```

endchar;

for the result



I hope that this description of the design and coding of a simple dingbat will encourage the reader to attempt one, or better yet, several. While the design of a complete font with METAFONT is a difficult and sometimes tedious process, the creation of a simple dingbat and some handy variations on it is not. It provides an enjoyable introduction to the use of METAFONT, and might just produce a dingbat that the reader can use to enhance and to personalize T<sub>E</sub>Xed documents. 

---

#### ERRATUM: "A Handy Little Font", Font Forum, TUGboat, Volume 10, No. 1

I regret that, inadvertently, I did something underhanded in my last *Font Forum* — to wit, I neglected to make the left and right braces visible in the code. My apologies to all who heeded the largish admonition at the end to "TRY IT", who were rewarded only with surly messages from METAFONT.

The macro for the whole handpointing character should read like this:

```
%Hand pointing right
def handpointing=
% define points for thumb and cuff
x1=x3=1/2[0,1/15w];
x2=x5=x4=x23=4/16w;
y1=y2=10/15[-desc,cap];
y3=y4=2/15[-desc,cap];
y5=6/7[y4,y2]; y23=1/7[y4,y2];
x6=9.75/16w; y6=y2;
x7=11.25/16w; y7=4/5[y23,y5];
x8=8.75/16w; y8=1/4[y7,y6];
x17=14.5/16w;
y17=9.25/15[-desc,cap];
% find a point at a certain height on
% the curve from z6 to z7
path dummyCurve; path dummyLine;
x.dummy=1/2[x8,x7]; y.dummy=y17;
dummyCurve:=z6{z5-z2}..z7..tension1.4..z8;
dummyLine:=z.dummy--z17;
z18=dummyCurve intersectionpoint dummyLine;
x16=x17; y16=y7;
```

```
x9=7/16w; y9=y8;
x10=6/16w; y10=2/5[y23,y5];
% find another point on the
% curve from z6 to z7
x.dummy2=x5;
y.dummy2=y16;
x.dummy3:=1/2[x8,x7];
y.dummy3=y.dummy2;
dummyLine:=z.dummy3--z.dummy2;
z12=dummyCurve intersectionpoint dummyLine;
% define points for curled fingers
x15=x14=x19=x22=1/3[x18,x17];
x13=x20=x21=x12;
y15=y16;
y13=y14=y15-(y17-y16);
y20=y19=y13-(y17-y16);
y21=y22=y20-(y17-y16);
% pick up pen and draw whole image
pickup pencircle scaled thinline;
draw z1--z2--z4--z3--cycle;
draw z5{(1,1)}..tension 1.5..z6
  &z6{z5-z2}...z7..tension 1.4..z8
  &z8{down}..tension3..z9
  &z9..tension 1.8..{left}z10;
draw z18--z17{right}..z16--z7;
draw z7--z15{right}..z14--z13{left}..z12;
draw z14{right}..z19--z20{left}..z13;
draw z19{right}..z22--z21{left}..z20;
draw z21{(-1,-1)}..tension1.5..z23;
enddef;
```

---

**Chess Printing via METAFONT and T<sub>E</sub>X**

Zalman Rubinstein  
University of Haifa

Every chess fan knows the pleasant difference between seeing an interesting chess position or a chess problem printed, and looking at the standard description of the pieces by means of an  $8 \times 8$  coordinate system a1 to h8. To help bridge this gap we have written a METAFONT-T<sub>E</sub>X program which enables one to print chess positions with ease, and to incorporate these positions with an arbitrary T<sub>E</sub>X output.

The implementation is based on the idea of dispensing with the creation of a separate chess board but rather in integrating the chess board with the chess pieces, thereby multiplying their number by two. We shall denote the chess pieces by the letters **K**, **Q**, **R**, **B**, **N**, **P**, respectively. The dark square will be designated by the letter **D**. In this notation the king appears in four shapes: **K**, **KD**, **DK**, **DKD**, meaning the white king on a white square, the white king on a dark square, dark king on a white square and finally the dark king on a dark square. Similarly for the queen, rook bishop, knight and pawn. It follows that the twenty four METAFONT designed pieces along with a single dark square suffice to assemble any chess position.

The shape of the chess pieces is based on simplicity rather than on artistic design at the present with the hope that improvements will be made at our METAFONT seminar this year.

It will suffice here to show the METAFONT code for the white pieces **K**, **Q**, **R**, **B**, **N**, **P** and the dark square **D** since all the other figures are easily deduced by METAFONT geometric and set theoretic operations such as **fill**, **unfill**, **draw**, **undraw**, **cullit**. The basic font was designed at 8pt size scaled 4000 (see illustration 1) on PC equipment. A smaller version scaled 3000 was also prepared (see illustration 2). The first approximation was set up on square paper with each box of size  $16 \times 16$  squares.

Following is the METAFONT code for the basic figures mentioned earlier.

```
beginchar("KING",8pt#,8pt#,0pt#);
h#:=8pt#; define_pixels(h);
pickup pencircle scaled 0.2pt;
pair w[];
w1=(2.5,0.5); w2=(2.5,1); w3=(0.5,11);
w4=(8,14); w5=(15.5,11); w6=(13.5,1);
w7=(13.5,0.5); w9=(6.5,14);
w10=(9.5,14); w11=(8,3);
```

```
w8=whatever[w6,w5];
w12=whatever[w2,w3];
ypart w12 = ypart w11 = ypart w8;
for i=1 upto 12: z[i]=h/16*w[i]; endfor
draw z9--z10;
draw halfcircle scaled(3*h/16) shifted z4;
draw z12--z8; draw z2--z6;
draw z1--z7; draw z11--z4;
draw z5--z4;
draw z1--z2--z3--z4--z5--z6--z7--cycle;
endchar;
```

```
beginchar("QUEEN",8pt#,8pt#,0pt#);
w1=(3,0.5); w3=(2,1); w4=(1,13);
w5=(4,2); w6=(8,13); w7=(7.5,14.5);
w8=(8,15.5); w9=(8.5,14.5);
w10=(12,2); w11=(15,13); w12=(14,1);
w14=(13,0.5);
ypart w2 = ypart w13 = 1;
w1-w2 = whatever*(w5-w4);
w14-w13 = whatever*(w10-w11);
for i=1 upto 14: z[i]=h/16*w[i]; endfor
draw z1--z2--z3--z4--z5--z6--z7--z8--
z9--z6--z10--z11--z12--z13--z14--cycle;
endchar;
```

```
beginchar("ROOK",8pt#,8pt#,0pt#);
w1=(3,0.5); w2=(3,1.5); w3=(4,1.5);
w4=(4,14); w5=(3,14); w6=(3,15.5);
w7=(5,15.5); w8=(5,14.5); w9=(7,14.5);
w10=(7,15.5); w11=(9,15.5);
w12=(9,14.5); w13=(11,14.5);
w14=(11,15.5); w15=(13,15.5);
w16=(13,14); w17=(12,14); w18=(12,1.5);
w19=(13,1.5); w20=(13,0.5);
for i=1 upto 20: z[i]=h/16*w[i]; endfor
draw z1--z2--z3--z4--z5--z6--z7--z8--
z9--z10--z11--z12--z13--z14--z15--
z16--z17--z18--z19--z20--cycle;
draw z3--z18;
endchar;
```

```
beginchar("BISHOP",8pt#,8pt#,0pt#);
w1=(5.5,0.5); w2=(10.5,0.5);
w3=(10.5,1); w4=(11.5,1);
w5=(11.5,5); w6=(8,14); w7=(4.5,5);
w8=(4.5,1); w14=(8,14.5); w9=(5.5,1);
w10=(7.75,5); w11=(8.25,5);
w12=(8.25,9); w13=(7.75,9);
for i=1 upto 14: z[i]=h/16*w[i]; endfor
draw z10--z11--z12--z13--cycle;
```

```
draw z1--z2--z3--z4...z5{up}...z6 &
  z6...z7{down}...z8--z9--cycle;
draw z9--z3;
draw fullcircle scaled (1*h/16) shifted z14;
endchar;
```

```
beginchar("KNIGHT",8pt#,8pt#,Opt#);
w1=(4.5,0.5); w2=(14.5,0.5);
w3=(14.5,4); w4=(12,10.5); w5=(5,15.5);
w6=(4.8,14); w7=(3.5,9); w8=(2,3.5);
w9=(4.5,4.5); w10=(8.5,5); w11=(5.5,2);
for i=1 upto 11: z[i]=h/16*w[i]; endfor
draw z1--z2 & z2..z3{up}..z4{z5-z2}..z5 &
  z5{z2-z10}..z6{down}..z7{dir 250}..
  (z8+(1.5,3)){down}..(z8+(3,0)){right}..
  z9{dir 30}..{right}z10 & z10{dir 240}..
  z11{z1-z10}..{down}z1;
endchar;
```

```
beginchar("PAWN",8pt#,8pt#,Opt#);
w1=(4,0.5); w2=(12,0.5); w3=(10.5,3);
w4=(11,4); w5=(9,13); w6=(7,13);
w7=(5,4); w8=(5.5,3); w9=(8,15.5);
for i=1 upto 9: z[i]=h/16*w[i]; endfor
draw z1--z2--z3--z8--cycle;
draw z3{right}..z4{up}---z5 &
  z5{dir 45}..z9{left}..z6{dir 135} &
  z6---z7{down}..z8{right};
endchar;
```

```
beginchar("chssqr",8pt#,8pt#,Opt#);
pickup pencircle scaled 0.3pt;
k:=8;
for i=1 upto k:
  draw (w/(2*k)*(2*(k+1-i)-1),w)--
    (0,w/(2*k)*(2*i-1));
endfor
for i=1 upto k:
  draw (w,w/(2*k)*(2*(k+1-i)-1))--
    (w/(2*k)*(2*i-1),0);
endfor
picture W,Z; cullit; Z=currentpicture;
clearit;
fill (0,0)--(0,h)--(w,h)--(w,0)--cycle ;
W=currentpicture;
addto Z also W; cull Z keeping (2,2);
currentpicture:=Z;
endchar;
```

To accommodate the chessfont a short  $\TeX$  macro enables printing the initial position in chess as follows:

```
\beginchess
\chessline\DR\DND\DB\DQD\DK\DBD\DN\DRD
\chessline\DPD\DP\DPD\DP\DPD\DP\DPD\DP
\whitechessline
\darkchessline
\whitechessline
\darkchessline
\chessline\P\PD\P\PD\P\PD\P\PD
\chessline\RD\N\BD\Q\KD\B\ND\R
\endchess
```

In a general chess position the white squares can be denoted by  $\backslash W$  or by  $\backslash \square$  and the black squares by  $\backslash D$ .  $\backslash whitechessline$  describes a horizontal chess line whose leftmost square is white, and similarly for  $\backslash darkchessline$ .

Obviously  $\TeX$  has the capability of producing a macro based on algebraic chess notation with only the pieces on board to be specified. We have not tried to do that.

It is to be noted that the program `chssqr` for the dark square is called in all pieces on dark squares as a subroutine. Because of the geometric design, in order to produce a new version of the twenty four chess pieces, it is only necessary to give the detailed programs of the six basic pieces with the rest following, as described earlier by transformations and set theoretical operations.

We shall conclude this note by listing the  $\TeX$  macro code and printing the illustrations mentioned earlier.

```
\font\chess = cheset scaled 4000
\font\chessm= cheset scaled 3000
\def\ifundefined#1{\expandafter
  \ifx\c#1\endcsname\relax}
\def\beginchess{\relax\begingroup
  \ifundefined{chess} \message{%
    Undefined font, replaced with cmtt10}
  \let\tt=\tentt
  \else \def\tt{\chess}\fi
  \tt\more }
\def\more{$$\vbox\bgroup
  \offinterlineskip\tabskip=Opt
  \hrule height 1pt
  \halign\bgroup
  \vrule width1pt
  ##&##&##&##&##&##&##&##&##&##&##
  \vrule width1pt\cr}
\def\endchess{\egroup
  \hrule height 1pt
```

This is the initial position in chess!

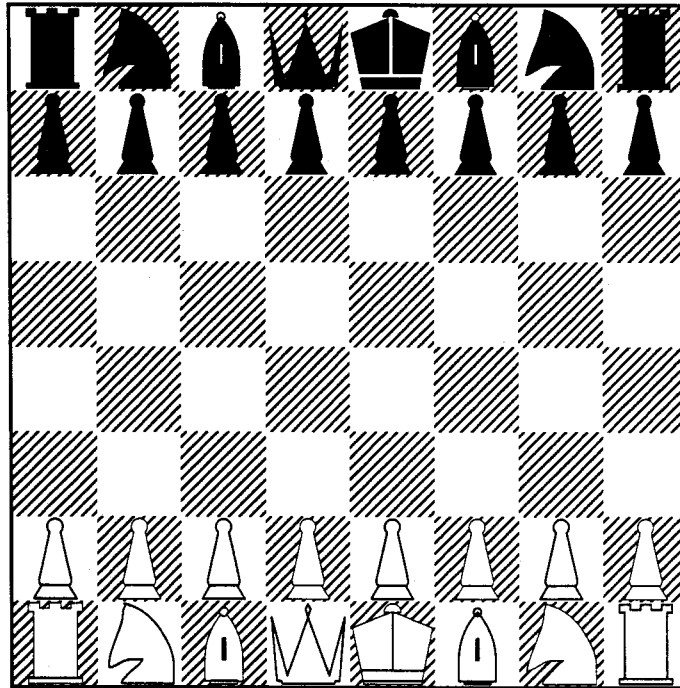
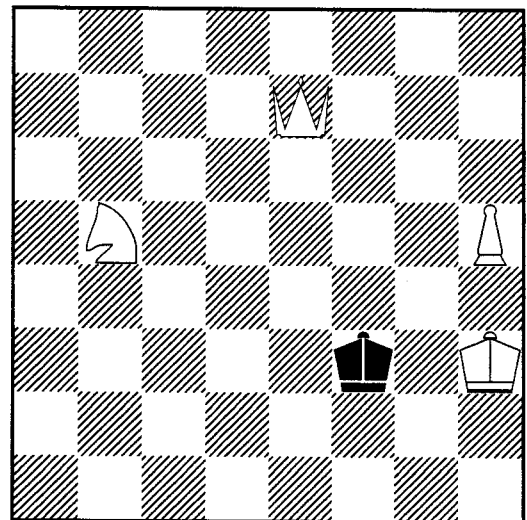


Illustration 1.

```

\egroup $$\endgroup\relax}
\def\whitechessline{&&D&&D&&D&&D&\cr}
\def\darkchessline{&D&&D&&D&&D&&\cr}
\def\chessline#1#2#3#4#5#6#7#8%
  {&#1&#2&#3&#4&#5&#6&#7&#8&\cr}
\chardef\K="01 \chardef\DK="02
\chardef\KD="03 \chardef\DKD="04
\chardef\Q="05 \chardef\DQ="06
\chardef\QD="07 \chardef\DQD="08
\chardef\R="09 \chardef\DR="0A
\chardef\RD="0B \chardef\DRD="0C
\chardef\B="0D \chardef\DB="0E
\chardef\BD="0F \chardef\DBD="10
\chardef\N="11 \chardef\DN="12
\chardef\ND="13 \chardef\DND="14
\chardef\P="15 \chardef\DP="16
\chardef\PD="17 \chardef\DPD="18
\chardef\D="19 \chardef\W="00
    
```

◊ Professor Zalman Rubinstein  
 University of Haifa  
 Department of Mathematics and  
 Computer Science  
 Mount Carmel  
 Haifa 31999 Israel  
 Bitnet: rsma407@haifauvm



Mate in three. Illustration 2.

My 64K chess computer solved it in twenty seconds.

---

## Guidelines for creating portable METAFONT code

Don Hosek

The T<sub>E</sub>X community is currently starved for new public domain meta-fonts and chances are that no matter how useless you think that some meta-font you may have created might be, there are at least forty people “out there” who would want it.

So with this the situation and chances that your code might find its way to operating systems vastly different than your own, I would like to offer the following guidelines to METAFONT designers for ensuring that their code can be run on other systems with a minimum of effort.

### 1 Internal documentation

METAFONT sources can go a long way and be transmitted in many forms. Just because you might send your source out in some encapsulated format (*e.g.*, `tar` or `arc`) doesn't mean that they will always be redistributed as such. More than once I've found myself with a file with an ambiguous name like `newzm.mf` and had no indication what it was part of or intended for. I would recommend that each source file you create contain the following information:

- The name of the file (this often can get lost).
- The last revision date. If you are modifying an existing METAFONT file, you should retain the old file's revision date and add your own along with a description of all changes made. This will allow easy updates to your file if the original file is later revised.
- The name of the package that the file is part of (*e.g.*, “CM Pica”, “Pandora”, etc.).
- Your name. This will make it easier for later users to track you down for complaints/suggestions/whatever. You may also want to include your current institutional affiliation and e-mail address as well.
- A brief description of the purpose of the file. This will make things easier on the individual who later attempts to follow the logic of your code.

None of these are *necessary* for allowing a METAFONT file to run other systems, but they will serve as an aid to users on other systems attempting to install your font.

Pierre MacKay has suggested the scheme shown in Figure 1 for this internal documentation (based on the file comments used by J. E. Pittman).

### 2 File names

The most important consideration when selecting filenames is to do your best to avoid file name conflicts. METAFONT's rule for selecting a file is to look for the indicated file in the current directory, then to look in `MFINPUTS` for the file. Personally, I believe that the current directory when METAFONT is run should be the one on which the METAFONT output will ultimately end up, so in that case, we are left with essentially a flat file space.<sup>1</sup> Thus it is essential to try not to have file name conflicts.

While in theory, this is a nice principle, it might be asked, “I can easily check my names against fonts that I have, but how can I be sure that I won't conflict with some odd font from someone else?” The short answer is that you can't. The longer answer is that it's possible to reduce the probability, simply by having all the files begin with the same initial combination of letters. For example, in an extra symbols font that I'm developing for use with internal fonts in the Xerox laser printers here on campus, I prefix each file used with the letters “cs” (for Century Schoolbook). While I can't know for certain that I've avoided all conflicts in this manner, chances are that name conflicts will not occur. As an added guard against conflicts, you might want to pick an additional arbitrary letter and tack it onto the file name to further guard against name conflicts (personally, I'm partial to “q”).

Another important consideration is the fact that all IBM systems (including PC's) have a file-name restriction of eight characters. Now, it's not strictly necessary to make all file names eight characters or less, but it would be helpful to at least guarantee that file names are unique to the first eight characters. The PC restriction makes this especially important as PC floppies are a convenient, inexpensive, and almost universally readable format for exchanging information. In addition, in a recent survey of T<sub>E</sub>X users in T<sub>E</sub>X<sub>M</sub>A<sub>G</sub>, over half the respondents used T<sub>E</sub>X on an IBM PC or compatible. Ignoring the eight character restriction can make your font inaccessible to a significant portion of the T<sub>E</sub>X community.

---

<sup>1</sup> Some operating systems, like IBM's VM/CMS give you a flat file space whether you want it or not; while one might be tempted to simply choose to ignore CMS as a METAFONT operating system, this is not feasible since the speed of IBM mainframes makes running METAFONT under CMS quite desirable—running METAFONT on Computer Modern fonts on an IBM 3081 took an average of 30 seconds per font!

```

% File:      MF Inputs U_Wash.mf
% Author:    Pierre A. MacKay
% Internet:  mackay@cs.washington.edu
% Bitnet:    mackay@cs.washington.edu
% Date:      November 27, 1988
%
% This is the University of Washington collection of |mode_def|s
% together with the macros to provide font-wide specials describing the
% |mode_def| that is used for each generated font, and the Xerox-world
% comments in the tfm file.  If a '?' is typed as the first response
% to the '*' prompt after this or a derived base file is loaded,
% a list of all current |mode_def|s will be given.
%
% This file follows a convention that has emerged in the discussion
% of |mode_def|s in TUGboat.
% 1. The print engine is identified wherever possible, rather than
%    the printer which incorporates that print-engine.
% 2. Because |mode_def| names may not contain digits, each digit is
%    given its full name, as in RicohFourZeroEightZero.
%
% WARNING: Some of the modes have never actually been tested

```

Figure 1: A model for internal METAFONT documentation

If you use a non-standard extension for any of your METAFONT files, you should take care that it is three characters or less, for the same reasons as the eight character limit above.

A file name should ideally consist only of the letters a-z and the digits 0-9. The first character of the name or extension should be a letter (some operating systems choke on file names beginning with numerals). When specifying a file name on an **input** statement, use all lower case; this will make life easier for the Unix people.

Finally, *never* include an explicit directory path on a METAFONT **input** statement. Since area names are necessarily system dependent, this guarantees that your code will not be portable.

### 3 MFT compatibility

MFT is a system for producing "pretty-printed" listings from METAFONT files; it was used in the production of Volume E of *Computers and Typesetting* and portions of *The METAFONTbook*. A complete description of MFT's capabilities and conventions is beyond the scope of this article,<sup>2</sup> but there are some simple things you can do to prevent MFT from blowing up.

<sup>2</sup> An MFT manual is in the works

- Use only a single percent sign on comments. MFT uses multiple percent signs to flag special handling code.
- Make sure that all comments are valid T<sub>E</sub>X input. If you refer to any METAFONT commands, variables, etc. enclose them in |...|.
- If you *must* comment out lines of code, either enclose the entire line in |...| as noted above or use four percent signs (%%%) to comment out the line.

### 4 Coding considerations

Most of the METAFONT code you write will be portable by default, but there are a few things that should always be taken into consideration:

- Never set *mode* inside the file. The proper way to invoke METAFONT is to say
 

```
MF \mode=whatever; input file
```

 This eliminates the need to specify *mode* inside the file. Similarly, *mag* should also not be specified in a METAFONT file.
- Keep the parameter definitions in a separate file from character definitions. While you might need a given font only at, say ten point roman, it's possible that someone else might want a nine point boldface of the characters you've designed. If you follow the model of Knuth's Computer Modern, this sort of modification will be much easier.



#### 4.1 Specifying dimensions

Always use sharped units when defining a dimension in your code and convert the sharped unit to pixels using one of the METAFONT commands listed on p. 268 of *The METAFONTbook*. I have encountered fonts which have specified things like **pickup pencircle** scaled 2 which will work fine on a dot matrix printer or even a write-black laser printer but looks awful on a write-white laser printer. What should have been done instead would be to specify some dimension such as *tiny*# which would later be converted using **define\_blacker\_pixels** into the appropriate pixel value for the output device.

METAFONT's sharped units provide a method for specifying units in a device-independent way. Rather than specify the widths of lines and other dimensions in terms of pixels, one first specifies units in terms of sharped units (you are given all of T<sub>E</sub>X's dimensions to begin with), then converts them with one of the macros listed below. Each is called in the form **define\_pixels**(*var\_one*, *var\_two*) where there can be as many variable names listed between the parentheses as necessary. For each variable name given, METAFONT will set its value according to a conversion into pixels from the corresponding sharped variables. In the example above, *var\_one* and *var\_two* would hold the pixel values of *var\_one*# and *var\_two*#.

**define\_pixels** Converts a sharped variable into pixels. This is done through a simple conversion. This should be used for variables which would not need any of the corrections described below. For example, a parameter used in calculating the widths of characters (such as Computer Modern's *u*#) would be converted into pixels with this command.

**define\_whole\_pixels** Converts a sharped variable into an integral number of pixels. This is normally used for variables which indicate the placement of certain points in the character.<sup>3</sup>

**define\_whole\_vertical\_pixels** Converts a sharped variable into an integral number of vertical pixels. This is used for the same sort of variables as **define\_whole\_pixels**, but takes into account any non-unit aspect ratio which may be used for the output device. This is generally used for vertical positioning while **define\_whole\_pixels** is used for horizontal positioning.

<sup>3</sup> For a complete discussion of why using integral values for various parameters is important, see Chapter 24 of the *The METAFONTbook*.

**define\_good\_x\_pixels** Converts a sharped variable into a value such that a pen drawn using the value as an *x*-coordinate will have its left edge on a pixel boundary. You must have a current pen selected for this to work. This is generally used for character sets (such as the one used in the METAFONT logo) where many of the characters are drawn using a single pen.

**define\_good\_y\_pixels** Converts a sharped variable into a value such that a pen drawn using the value as the *y*-coordinate will have its top edge on a pixel boundary. This is the vertical analogue to **define\_good\_x\_pixels**.

**define\_blacker\_pixels** Converts a sharped variable into pixels adding METAFONT's *blacker* to the value obtained. This should be used for a variable which will determine the width of lines drawn or pens used. This is a very important definition since without it, METAFONT's **mode\_def** convention is almost useless.

**define\_whole\_blacker\_pixels** Converts a sharped variable into an integral number of pixels taking METAFONT's *blacker* into account. This should be used for variables which will determine the width of lines drawn or pens used which should be set to an integral value.

**define\_whole\_vertical\_blacker\_pixels** Converts a sharped variable into an integral number of vertical pixels. This has the same relationship to **define\_whole\_blacker\_pixels** as **define\_whole\_vertical\_pixels** has to **define\_whole\_pixels**.

**define\_corrected\_pixels** Converts a sharped variable into a pixel value after taking into account the curve overshoot parameter (METAFONT's *o\_correction*). This should be used on variables which give the overshoot for a curved portion of a character (*e.g.*, the bottom of "U"). *The METAFONTbook* has details on when this is appropriate.

**define\_horizontal\_corrected\_pixels** Similar to **define\_corrected\_pixels** but does the rounding for a horizontal value.

The METAFONT logo font (which is described throughout *The METAFONTbook* and listed in its entirety in Appendix E of that work) is a good simple example to see how these different METAFONT commands are used.

#### 4.2 Compatibility with Computer Modern

Unless your font is designed explicitly for use with some non-Computer Modern font (*e.g.*, extra symbols for use of T<sub>E</sub>X with a printer-resident font), it

is probably a good idea to plan your type so that it is visually compatible with Computer Modern. You will probably also want to follow the existing  $\TeX$  coding schemes (except for odd fonts such as an astronomical symbols font) as well. These practices carry with them several benefits:

- By following existing coding schemes you make it easier to achieve compatibility with existing  $\TeX$  macros.
- Visual compatibility with Computer Modern allows you to use CM fonts for things such as typewriter type and math if you so choose. In addition, if type “A” is visually compatible with Computer Modern and type “B” is visually compatible with Computer Modern then types “A” and “B” should be visually compatible with each other.

The primary objective when striving for “visual compatibility” is to guarantee that the characters should align well with Computer Modern. At the very least, baselines of characters should match well. To allow use of Computer Modern math fonts with your typeface, the weights of the characters should roughly correspond to the weights of corresponding characters in CM.

As an example, consider Figure 2 which mixes Computer Modern and Concrete together in several contexts. These two typefaces have a roughly corresponding character grid, but the difference in weights produces an odd mixture when the two are combined.<sup>4</sup> Overall, the samples above give some indication of the flexibility obtained by striving for compatibility with Computer Modern.

---

Concrete and Computer Modern Roman will not  
mix well.

Concrete and Computer Modern typewriter type  
will blend somewhat better.

Concrete does not produce optimum results with  
 $\Gamma + k\alpha = 0$  Computer Modern math.

**Figure 2:** Mixing Concrete and Computer Modern in some different contexts.

◊ Don Hosek  
3918 Elmwood  
Stickney, IL 60402  
Bitnet: u33297@uicvm

---

<sup>4</sup>In fact, as was explained in *TUGboat* 10(1), these fonts were designed for use with the Euler math fonts.

## Graphics

### Integration of $\TeX$ and Graphics at the Pittsburgh Supercomputing Center

Phil Andrews

#### Our Graphics Environment

The Pittsburgh Supercomputing Center is one of five NSF national Supercomputer Centers established to help scientific researchers. Our main machine is a CRAY YMP8/24 (8 processors, 24 million 64 bit words of memory) running UNICOS, the CRAY version of UNIX. We also have a large number of DEC machines running both VMS and ULTRIX, other general purpose computers such as a HARRIS HCX/UX and special purpose computers such as an ARDENT TITAN. Presently we have approximately 1000 users, the great majority of whom are remote and communicate with our center over networks such as the Internet.

The center was established in the summer of 1986 in the enviable position of being able to design our overall graphics system from scratch, having no compatibility requirements. We decided to standardise on the CGM (Computer Graphics Metafile) format for picture storage and acquired only graphics packages supporting that format, e.g., DISSPLA from CA-ISSCO, DI-3000 from Precision Visuals and the NCAR Graphics package from the National Center for Atmospheric Research. If possible we adapted other packages, such as MOVIE.BYU, to produce CGM format files.

While in 1986 CGM was an emerging standard for picture storage, it is now both an ANSI and ISO standard and is solidly ensconced as the preëminent format for the description of two-dimensional graphical data. There are several interpreters available for the display of CGM files on various output devices, but in general they can only display a small subset of the CGM elements and are specific to the CGM files produced by that vendor's graphical packages. In addition they may be proprietary, and, if purchased, we could not redistribute them to our remote users. In order to mitigate the problems of network access, we encourage our users to generate CGM files on our machines, and then ship the CGM output home to their host machines where it can be viewed in a more interactive manner.

For these reasons, and because we wanted to support any available output device, we decided to write and maintain our own CGM interpreter, called

GPLOT for General PLOTting program. GPLOT now processes all of our users' CGM files, independently of their origin, and is used in-house for the production of video animations (approximately 10 hours of animations in the last 12 months) which we mail to users. We support numerous output formats, including PostScript and QMS (QUIC) laser printers, several types of workstations (via UIS, X11, or CGI interfaces), numerous Tektronix and other terminals, GKS devices and video output via a Peritek frame buffer. We are continuously updating the number of devices supported. GPLOT will accept either binary or clear text format CGM files and will convert between the two.

#### Integration with $\TeX$

In designing the GPLOT system, I wanted to address one of the outstanding problems in computer related information interchange: the problem of text and graphics integration. I chose the  $\TeX$  typesetting system for this purpose, partly because of its popularity and capabilities, and partly because of familiarity. In the late 1970's I ported first the old Pascal version and then the WEB version to both TOPS-10 and VMS and wrote DVI processors for Versatec and Tektronix output devices, and (later) for QMS laser printers.

I removed the DVI interpretation part of that program (GTEX), together with the font manipulation system, and integrated it into emerging CGM software to form our GPT system. Although GPLOT and GTEX are separate commands, they are part of a completely integrated system with over 90% of code in common. Any CGM file processable by GPLOT can be included in any  $\TeX$  file by GTEX, and GPLOT uses the standard  $\TeX$  font files (in PK format) for the textual parts of CGM files on output devices with limited textual capability.

Naturally the  $\TeX$  interface to the CGM graphics routines is by way of a `\special` command, but many different formats are possible. With over 1000 users from differing backgrounds, documentation and training can be a significant problem, and, with this in mind, I decided to make the command format for the `\special` command identical to what the user would type at the command level. That is, if (under VMS) the user would type

```
gplot/dev=ps/pag=3/x_size=4.5/y_size=4.5 foo
to instruct GPLOT to process the CGM file
foo.cgm, extract the third page only and produce
output for a PostScript printer scaled to fit
```

a 4.5 by 4.5 page, then inside a  $\TeX$  file the corresponding `\special` command would be

```
\special{gplot/pag=3/x_size=4.5/y_size=4.5 foo}
```

Note that the `\special` command itself is device independent, the output device being specified on the GTEX command line that caused the interpretation of this particular DVI file. Any device specification inside a `\special` command will be ignored. The current page position in the DVI file will become the origin for the included graphics page. Any number of pictures from any combination of files can be included in an individual  $\TeX$  page. However the effect of overlays is completely device specific.

### **$\TeX$ processing specifics**

GTEX and GPLOT use the same command line parser, with identical sets of options and supported devices; if the device can use downloaded fonts (e.g., PostScript or QUIC) then I use that capability (only the required characters are downloaded). Expanded versions of the character descriptions from PK format files are cached either internally in memory, or externally in an indexed file. For any  $\TeX$  file of several pages or larger, the font processing is normally a small part of either the CPU or I/O requirements, and I do not attempt to use native fonts for  $\TeX$  output. If the device cannot download fonts then I convert character references to cell array calls (the CGM raster operation). Rules naturally map to CGM rectangle calls. One interesting result of this mapping is that GTEX can convert DVI files into CGM files; these CGM files, however, will be partly resolution-specific with each character represented by its bitmap. A proposed addendum to the CGM standard adds segments (recallable descriptions), and as soon as this becomes official I will use this operation rather than cell arrays for character representation. There are many devices that do not support downloadable fonts but do have segment support, and of course the associated CGM files will be much smaller.

### **Problems**

GPLOT and GTEX are wholly in the C programming language and run under both UNIX and VMS, but with different command interfaces. That means that presently the `\special` command syntaxes are also different under each system, an unacceptable format. As our users become more accustomed to UNIX I hope to move away from the VMS format for `\special` commands, supporting the UNIX format on both systems. However I plan to continue to

support the VMS format under VMS for our users who are unconcerned with portability.

Many CGM files use colour tables for their output, then use indices into this table to designate colours. When two such pictures, with distinct colour tables, are included on a single  $\TeX$  page there may be clashes, depending on the capabilities of the output device. I intend to eventually do an automatic conversion from indexed to direct colour in these cases. The difficulty is in deciding when this is necessary.

### **$\TeX$ -specific graphics**

In some cases what is really required is the capability to process simple graphics at runtime, rather than the inclusion of complex preprocessed graphics. In this case the difficulty lies in organisation rather than implementation. If there is no requirement for interaction with the location of  $\TeX$  elements then a simple clear text CGM file can be written and included. For more complex requirements I have allowed (for experimentation purposes) a simple `\special` command that allows access (at runtime) to the capabilities of any CGM element processor. This can be used to change text colour, produce lines linking  $\TeX$  elements, fill polygons, etc. What is needed here is some set of simple `\specials`, preferably community wide. However it should not be thought that some such set can satisfy all graphics requirements. Many graphics packages have more manpower investment than the entire  $\TeX$  system and cannot be easily simulated.

### **Availability**

We are presently distributing GPLOT/GTEX freely via Internet (we don't write tapes), although it is copyrighted and we request that you do not redistribute any modified, spindled, folded or mutilated copies. Nor can you sell it, include it any package for sale, etc. For further information send mail to `ANDREWS%CPWSCB@CLIPR.PSC.EDU`.

◊ Dr. Phil Andrews  
Pittsburgh Supercomputer Center  
Mellon Institute  
4400 Fifth Avenue  
Pittsburgh, PA 15213  
`andrews%cpwscb@clipr.psc.edu`

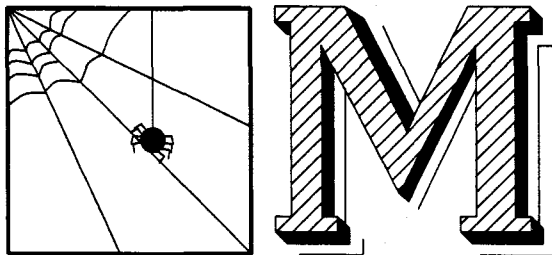
# METAPLOT

## Machine Independent Line Graphics for T<sub>E</sub>X

Patricia P. Wilcox

Last winter my husband and I set out to use AmigaT<sub>E</sub>X to document a collection of FORTRAN mechanical engineering programs. We had a stack of drawings to include: some generated directly by the FORTRAN programs, and some created using the Aegis Draw 2000 program on the Amiga.

It would be a big help to be able to print those drawings with T<sub>E</sub>X! I decided to give it a try, using METAFONT to do most of the work, and within a few days I had T<sub>E</sub>X drawing pictures like these two:



From that beginning has grown a set of METAFONT and T<sub>E</sub>X macros collectively referred to as "METAPLOT". Here's how it works. The METAPLOT macros enable METAFONT to simulate a line plotter, so that it can turn a (suitably pre-formatted) line plotter command file into a picture variable. The picture is chopped up into rectangular tiles which are shipped out as "characters" that can be typeset by T<sub>E</sub>X.

This is not a particularly efficient way to do things; there is extra overhead because METAFONT runs as an interpreter, and even more overhead due to the diabolical deviousness of METAFONT's mental processes. METAFONT run time for a single drawing is likely to be several minutes. On the other hand, this approach is virtually system and device independent. Slowness notwithstanding, METAPLOT is proving to be a simple and useful way to add line drawings to T<sub>E</sub>X documents.

We present an account of our journey on the road to illustrated T<sub>E</sub>X, in the hope that the reader will find some interest therein, and perhaps benefit from advance knowledge of some of the landmarks and pitfalls to look for in his own excursions into the wilder regions of METAFONT and T<sub>E</sub>X.

## Scope of the Project

Our graphics needs were modest, compared with some of the ambitious work being undertaken by other members of the T<sub>E</sub>X community. For one thing, there was no need to worry about line color. If colored printing should ever be required (unlikely!) we will use the CAD software to separate the colors; what a human printer wants to see is a separate *black* line drawing for each color.

I also chose to ignore dot fill patterns, at least initially, because METAFONT is terrible at dealing with closely-spaced fill patterns over large areas. This is because METAFONT encodes edges, not points; a dot fill pattern has a lot of edges! (I can envision some desperate workarounds for this problem, but basically, if you need to do this sort of thing, you should probably be using a PostScript drawing program with a PostScript output device.)

METAPLOT does not attempt to process half-tone photographs, which have much the same problems as dot fill patterns. Besides, I have enough trouble getting acceptably printed photographs when I am working with an experienced printer who uses a superb copy camera and metal printing plates on a high-precision press! Most T<sub>E</sub>X output devices just aren't good enough yet to print half-tones cleanly.

By limiting our scope to the representation of black lines and solid areas, we thought to have a useful project that could be accomplished quickly, so that we could get back to the original task of publishing documents.

## In Search of Graphics Standards

My first step was an informal look at standardization (or lack thereof) in the engineering graphics field; METAPLOT could be much more generally useful if it did not depend on a drawing format specific to one drawing program or one computer manufacturer.

I looked first at standards for the logical representation of graphical objects. The possibilities were IGES (a standard adopted by the U.S. in 1981) [1], GKS (see [1] as well as discussion by Bart Childs et al. in the April TUGboat [2]), and DXF, the AutoCAD drawing exchange format [3], which is something of a *de facto* standard in the industry. Quite a few of the common CAD programs attempt to support IGES, DXF, or both, although what I have been hearing is that you can expect about an 85% success rate in transferring a "standard" drawing between unrelated software packages with IGES or DXF, and that the standards change from week to week. Not good enough!

Although Bart Childs *et al.* (reference [2] again) say that “Most vendors deliver reasonable support for a GKS environment or it is available from third party vendors for common systems,” at the time I was designing **METAPLOT** it was not evident that the GKS standard was supported by *any* CAD programs for any of the common personal computers. It may be that Dr. Childs is talking about large mainframe computers.

The final blow is that not one of the four computer aided drafting programs I use on the Amiga is smart enough to understand IGES, DXF, or GKS.

The next place to look was on the output side: display standards. There are two major philosophies at work in the structured graphics software out there today. For want of a better term, call them “traditional” vector graphics and PostScript graphics. PostScript is rapidly winning the field, because it’s more powerful than straight vector graphics, and vector graphics capabilities can be handled as a subset of the functions supported by PostScript.

In a traditional structured graphics program, lines are drawn by moving a pen or an electron beam along the shortest path from here to there. There are no true curves, only straight-line approximations. If fill patterns are used, they have to be something that can be drawn with line segments.

PostScript graphics programs allow you to generate curves from their Bezier control points and fill areas with arbitrarily fine and complex pixel patterns. Many **T<sub>E</sub>X** implementations, including **AmigaT<sub>E</sub>X**, already have PostScript “\special” commands which allow you to integrate PostScript graphics with **T<sub>E</sub>X** documents for output to a PostScript printer. There’s just one little problem — if you speak PostScript, you can speak only to something that understands PostScript. Since I want to print **T<sub>E</sub>X** documents on “dumb” lasers and dot matrix printers, and I have considerable investment in traditional vector graphics software and data files, PostScript will not work for me, yet.

Things are changing rapidly. It is encouraging to hear about the good work of people much braver than I who are working on PostScript interpreters like the one described in “News from the **VORTEX** Project” in the April **TUGboat** [4]. If such interpreters were universally available, the task of importing vector plot files into **T<sub>E</sub>X** documents would be reduced to writing a simple program to translate vector commands to PostScript commands.

But, let’s face it, PostScript was designed as a “write-only” standard. It’s straightforward to write

a program that produces PostScript output, but tricky to write a program that does a 100% correct job of interpreting PostScript code and turning it back into a bit-mapped image to drive a non-PostScript device. The “real” PostScript exists only in the microcode of PostScript display devices, and is not generally available to developers. The problem is exactly analogous to what we would face if we were asked to re-create **T<sub>E</sub>X** and **METAFONT** from an external description of their behavior, without the benefit of access to the original code and without “trip” and “trap” tests to ensure adherence to the standard.

Instead of waiting around until there was a PostScript interpreter that could do my work for me, I looked for something less elegant, but simple and general, along the lines of the “standard display file format” described by David F. Rogers [5] in the last **TUGboat**. It didn’t take long to find what I was looking for. A sort of lowest common denominator between all of these CAD software packages is that they all know how to drive pen plotters, using a very small set of graphics primitives. This “standard” has the great advantage of being enforced by a machine. Deviations from standard are punished by the fact that the plotter simply won’t work!

If you look at Hewlett-Packard Graphics Language (**HPGL**), which is understood by HP plotters (and a lot of other plotters on the market) you find the following set of actual drawing commands:

Pen motion:

PU pen up  
 PD pen down  
 PA plot absolute  
 PR plot relative  
 CI circle  
 AA arc absolute  
 AR arc relative  
 LB label (draw text)

Line specification:

LT line type (dot/dash pattern)  
 SP select pen color

-----  
 Special purpose commands, mostly  
 for graphs & pie charts:

FT specify fill pattern  
 [type [,spacing [,angle]]]  
 EA,ER,EW outline rectangle/wedge  
 RA,RR,WG shade rectangle/wedge  
 XT,YT draw X and Y tick marks

along with a collection of auxiliary commands to do things like plot scaling and plotter initialization and cleanup. The specialized commands below

the dashed line do not really belong in a standard command set. Of the commands above the line, if we omit the “relative” commands, we haven’t lost any functionality. This leaves, for a “standard” set of line plotter commands:

```

Pen motion:
  PU pen up
  PD pen down
  PA plot absolute
  CI circle
  AA arc absolute
  LB label (draw text)
Line specification:
  LT line type (dot/dash pattern)
  SP select pen color

```

Could I omit anything else? Looking closely at the actual HPGL plot commands used by CAD software, you’ll find that not everything in the list is required by every graphics application. Ignoring plotter setup and scaling, the three programs I use on the Amiga (Aegis Draw, IntroCAD, and mCAD) use just four commands: “move”, “draw”, “line type”, and “pen color”. Generic CADD (on the IBM PC) makes do with even fewer commands; it uses only “move”, “draw”, and “pen color”.

However tempting it was to pare down the list further, I had one application (John’s FORTRAN plot package) that was going to need CI and LB commands, and I later found another (VersaCAD) that also used CI. Better leave them in.

Plot scaling would be done by scanning the data (after rotation) for min and max  $x$  values, and multiplying all coordinates by the ratio of printed width (specified by the user) to width of the data ( $x_{max} - x_{min}$ ). This is scaling from a printer’s point of view, where the important final dimension is column width.

#### METAPLOT — Initial Implementation

Including the commands in the standard command set didn’t mean I had to implement them right away. Color was at the bottom of my list; dashed lines were near the bottom; label was too complicated to deal with on the first pass. I chose to start the implementation of METAPLOT by writing METAFONT macros analogous to plotter “move” and “draw” routines.

You may notice that something is missing. METAFONT is not very good at character string manipulation. Surely we don’t want to write an HPGL language interpreter in METAFONT! How do we get plotter commands translated to a form that METAFONT can understand?

There were three answers to that. One of the first things we did was to write a version of the FORTRAN plotting routines with output in the form of METAPLOT macro calls instead of plotter commands. This took care of the first stack of drawings.

Translating the second category of drawings (plots from the CAD program) depends on a sneaky trick with Aegis Draw—watch closely! Aegis Draw has the virtue of allowing the user to supply his own plotter configuration file containing an initial string, a separator, and a terminator for each plot command. I created a configuration file defining a rather strange imaginary plotter called “META”. When plotting to META, Aegis Draw emits METAFONT macro calls instead of physical plotter commands. Here’s a small plot in HPGL, with the equivalent META plot commands as first implemented back in January:

HPGL:	META:
-----	-----
IN;DF	beginplot;
SP 1	pencolor(1);
LT 5	linetype(5);
PU;PA 100,100	moveto(100,100);
PD;PA 200,100	drawto(200,100);
PD;PA 150,167	drawto(150,167);
PD;PA 100,100	drawto(100,100);
IN;DF	endplot;

Later on, I changed the most frequently-used META plotter commands to more efficient 2-character codes, and added an explicit “-1” for each unused HPGL parameter. (Line type has an optional second parameter indicating pattern size.) The same drawing, revised, looks like this:

beginplot;	% EXPLANATION:
sp(1);	% pen color
lt(5,-1);	% pattern,spacing
pu(100,100);	% move to x,y
pd(200,100);	% draw to x,y
pd(150,167);	
pd(100,100);	
endplot;	

The third way to convert plot commands to META commands is a preprocessor called VGtoMF, which is just now (May) becoming a reality. I’ll save VGtoMF to talk about later, because a lot of things happened to METAPLOT between January and May.

Once the META plot file exists, we need a METAFONT driver file to generate the plot font. This looks a lot like any METAFONT font generation file, with a few extras needed for handling line

drawings. (This sample anticipates the tile/mosaic scheme described in the next section.)

```

mode_setup;
font_size .80 in#;
numeric current_char;
input plotmacs;      % METAPLOT macros
print_width:=1.5;    % Inches! Using
max_tile_width:=.80; % dimensionless
max_tile_height:=.80; % numbers here is
                    % a design flaw.

print_rotation:=-90;
first_letter_code:=1;
plotter_pen_weight:=7;
    %pen weight in plotter steps
mosaic ("myplot")(first_letter_code);
    %characters generated here!

font_slant 0;
font_normal_space Opt;
font_normal_stretch Opt;
font_normal_shrink Opt;
u#=.1in#;           % These values
font_x_height 5u#;  % don't mean much
font_quad 2u#;      % for a line plot
font_extra_space 2u#; % ...
bye.

```

The resulting font could be used in  $\text{T}_\text{E}\text{X}$  by explicitly typesetting the characters:

```

\font\plotfont=myplot50
....
{\offinterlineskip\plotfont
 \centerline{\char1\char2}
 \centerline{\char3\char4}
}

```

or by invoking the “\plot” macro in `plotutil.tex` (the third component of the **METAPLOT** package):

```
\offinterlineskip\plot 1 {myplot50}
```

### The Evolution of Modern **METAPLOT**

**Dealing with Finite Memory.** The biggest problem with our first test plots was that they weren't big enough. They were small for two reasons: METAFONT memory limitations and device driver limitations. A complex plot (with lots of vertical structure) will run out of memory sooner than a very simple plot, but in any case, it doesn't make sense to expect to keep a bit image of an entire plot in memory at one time.

(Another reason for keeping character sizes reasonably small is that some printers are limited to characters 255 pixels on a side; this does not happen with my DeskJet printer.)

It is not very easy to increase memory allocation for Amiga**METAFONT**—it uses the maximum memory addressable by 16-bit pointers. Adding more memory would require doubling the size of all address pointers.

Discussing Turbo**METAFONT** in the last TUGboat [6], Richard Kinch says, “We do not now see the need to include the virtual memory simulator in the Turbo**METAFONT** programs ... the enterprise of generating fonts does not seem to encourage the use of enormous macros or tables ...” **METAPLOT** may provide an incentive for including virtual memory in **METAFONT**, since **METAPLOT** could process a large picture in a fraction of the time it takes now, if the complete picture could be kept in memory at once.

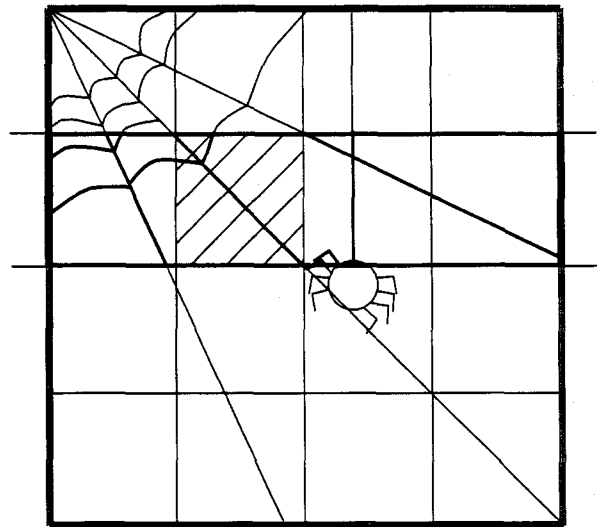


Figure 1. Mosaic tile layout.

Here's our initial solution for dealing with limited memory (Figure 1). A one-dimensional clipping scheme is used, since 1-D clipping is faster than true 2-D clipping. Two new variables (`max_tile_width` and `max_tile_height`) were added to the font generation commands to specify how the picture should be subdivided. The drawing is divided into  $n$  horizontal strips, the plot file is scanned  $n$  times, and on each pass, all lines are clipped against the top and bottom of the current strip (remembering to add half the pen width at the top and subtract it at the bottom, so as not to lose the edge of a line whose center just misses the box limits); after clipping, any visible parts of the path are added cumulatively to a picture variable. Then (and this part is pretty fast) **METAPLOT** moves from left to right across the strip, and-ing the picture with a



black box exactly the size of one tile, and shipping out the resulting piece of the puzzle as a character.

Note that the clipping is used only to reduce the amount of memory used to store the picture. The final character edges are determined by the and-ing operation.

I had assumed that arbitrarily large pictures could be processed by reducing the tile height, thus requiring fewer square inches of picture to be stored at one time. This works, up to a point, but there is some tile height for which the method breaks down. The figures in this article were well within AmigaMETAFONT's memory limits, even at TUGboat's 723 dpi resolution, but some of our FORTRAN-generated plots exceed memory capacity at large size or high resolution.

**Streamlined plots.** Having, after a fashion, resolved space problems, I started looking for improvements in the time domain. A 2:1 performance improvement resulted when I stopped culling the picture variable after each path was added. (One wonders what other easy speed-ups might still lurk in the code ...) One thing that would certainly make things faster and reduce the size of plot files would be to dispense with line-segment approximations to circles, curves, and filled areas, and let METAFONT generate the curves mathematically.

It was frustrating—here was METAFONT, which could solve all of our problems, acting like a dumb line plotter. How could the CAD program send circle, spline, and fill commands to METAFONT? Well, we weren't using the pen color command for anything ...

And here was born the first of several "graphical escape sequences". Let's say (assigning arbitrary colors to pen numbers for purposes of discussion) that we use pen 1 (black) for plain lines, pen 2 (blue) for "fill", pen 3 (green) for "filldraw", and pen 4 (red) for "erase". Then if we hop over and appropriate one of the line types on the DRAW menu, and, by convention, call it a circle-drawing pen, and use another spare line type for a spline-drawing pen, we should be able to transmit some fairly useful requests to METAFONT.

**Circle-drawing pen conventions:** (These are METAFONT near-circles and super-ellipses)

- A rectangle drawn with the circle pen is converted to the ellipse bounded by the rectangle.
- A single line segment defines a circle—leftmost point in print coordinates is the center, rightmost point is on the circumference. (Don't count on your CAD package not to flip lines

end-for-end. Saying "first point is center" didn't work at all well.)

- Alternatively (thanks to Bill Hawes for this idea), use a square box to specify a square ellipse, which is, of course, a circle.
- A triangle (which is a 4-point path with beginning and end superimposed) is used to represent an arc—point 1 = point 4 = center of arc. Pick the shorter of the first and last sides; this will be the radius. The arc is drawn counterclockwise around the circle, from shorter side toward longer side.

**Spline pen conventions:** A splined path consists of  $3N + 1$  points, defining Bezier curves, four points per segment, with the center two points of each segment being control points. (It may take a bit of practice to develop the knack of defining a curve by its control points, if your CAD program doesn't display the curve.)

Pen type and path type are defined so that they can be paired up in any combination; you can, for example, draw a green ellipse, and METAFONT will use filldraw to add the ellipse to the picture; or draw a red circle and METAFONT will erase that circular area of the drawing.

This is a bizarre-looking way to enter data! What you see on the screen has very little relationship to the METAPLOT picture you are creating. It helps to use brown (invisible to METAPLOT) to draw a temporary copy of a line-segmented ellipse or arc as a visual indication of the figure symbolized by the rectangle or triangle you're sending to METAPLOT. To keep plot files small, erase the brown temporary copies before writing out the plot for META. (I haven't told you about brown. After trying out the red/green/blue lines, we added pen 5 (brown) for lines that will be invisible to METAFONT except for computing min and max  $x$  and  $y$ —good for bounding boxes and construction lines. And we added pen 6 (purple) for half-weight lines and pen 7 (orange) for half-weight filldraw.)

Figure 2 demonstrates the use of graphical escape sequences to fool Aegis Draw into generating an assortment of things that are "impossible" to do with Aegis Draw.

Now one last thing would be *really* useful, and that is a way to graphically specify typesetting commands with the CAD software, and have METAFONT pass the typesetting requests along to T<sub>E</sub>X for final realization. OK, let's see ... to be a legal splined curve, a path must consist of  $3N + 1$  points. A rectangle is a 5-point path, so it can't be a spline. We'll specify the position of a typeset

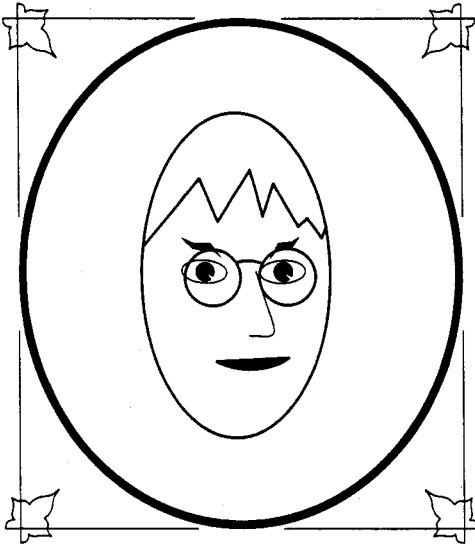


Figure 2. "Self Portrait"  
Demonstration of circles, arcs, and splines.

label by drawing a brown rectangle with the spline pen. We can start at the lower left corner for left-justified text, start at the lower right corner for right-justified text, and start at one of the top corners for text centered in the box.

We have METAFONT compute the corner coordinates for each text box (in true inches on the printed page, down and across from the upper left hand corner of the plot), and write a T<sub>E</sub>X typesetting command to the METAFONT log file, complete with position information and a label number to print on the first draft. After the user sees the first T<sub>E</sub>X draft of the plot, he can replace label numbers with the appropriate text and move labels by adjusting  $x$  and  $y$  coordinates in the T<sub>E</sub>X file, without incurring the overhead of running METAFONT a second time.

While we're on the subject of typesetting, I should mention that complete typesetting information for each mosaic of plot characters has been included in the typeface itself, in the form of `\fontdimen` parameters (thanks to Tom Rokicki for nagging me to do this). METAFONT writes the one-line macro call that reassembles the plot mosaic, along with the rest of the typesetting information, in the METAFONT log file. After METAFONT has created your plot type face, extract the typesetting commands from the METAFONT log file and insert them in your T<sub>E</sub>X file at the point where you want the picture printed, and you're done.

Using little invisible typesetting boxes, I did the typesetting for Figure 3 at least ten times as

fast as I could have done it with pencil and ruler. What a relief!

(Note for dingbat enthusiasts: the `\fontdimen` parameters now include line weight in printer coordinates, so that fancy METAPLOT characters can be joined with T<sub>E</sub>X rules of the right thickness.)

### VGtoMF: A Universal Vector Graphics Interface?

According to what I've told you so far, the Aegis Draw program on the Amiga and a mysterious FORTRAN program are the only programs in the world that can generate command files for the META plotter. If you look at the sample plot listing, you can see that it would not be very difficult to convert HPGL commands to META format using nothing more than a text editor: but this is hardly an elegant solution!

The trick to making METAPLOT portable to all systems is to have a nice simple easily ported C program that reads plotter configuration files describing 1) the syntax of your existing plot file and 2) the command syntax of the plotter you want to convert to. This is intended to convert *from* something else *to* META, but in theory it ought to be able to convert from any plotter to any other plotter.

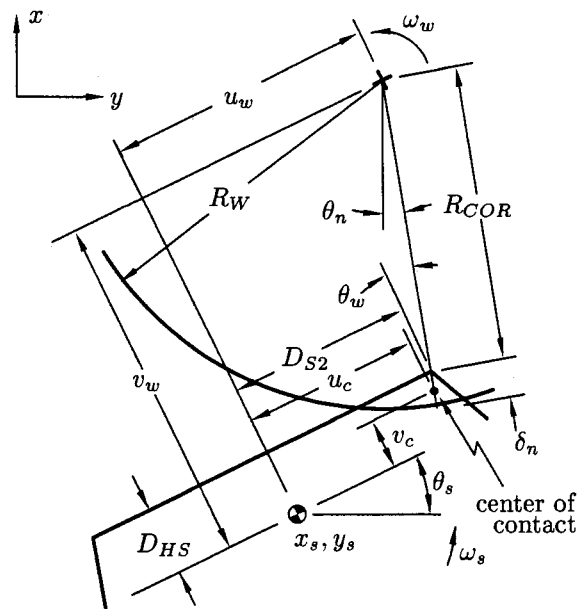


Figure 3. A typical engineering drawing, using METAPLOT typesetting boxes to position labels.

There are a couple of traps here that I should warn you about. Remember the 4095+ limit on numeric values in METAFONT? Given that coordinates in HPGL are usually written as integers, this creates a rather narrow range of plot coordinates where the plot program is not losing accuracy and **METAPLOT** is not blowing up with illegally large numbers in its transforms. A related issue is that, in the present version of **METAPLOT** at least, the  $x$  and  $y$  dimensions of a picture, measured in printer steps, may not exceed 4095. This is not a problem at low resolutions, but it would limit picture size on a 2000 dpi printer to just over two inches.

Furthermore, mCAD, my favorite shareware graphics program on the Amiga, needs a little help in fixing up its  $x$ -to- $y$  aspect ratio. And, as I discovered in preparing the drawings for this article, if you wish to plot at high resolutions, you had better be prepared to center your plot directly over the  $x,y$  origin, again to prevent numeric overflow of transforms. Things would be much more comfortable if coordinates were decimal numbers pre-scaled to reasonable limits: say  $x$  and  $y$  coordinates ranging from about  $-1000.000$  to about  $+1000.000$ .

The HPGL language permits a path to be represented by a single draw command followed by a list of  $x,y$  pairs—yet another syntactic variation that the conversion program must be able to handle.

A quick look at some HPGL output from the Macintosh version of VersaCAD revealed that it was using CI (circle) commands as well as the move, draw, pen color, and line type commands we were expecting. OK, we'll add CI to the list of required commands.

It's becoming apparent that VGtoMF has to be more than the simple string-substitution editor I originally set out to write! Required functions are:

- Command string substitution
- Aspect ratio correction
- Coordinate transformation
  - Translation
  - Scaling
- Special work-arounds

It's just about working now, with code to take account of all the quirks I know about, but it's clear that for every new CAD program someone wants to use with **METAPLOT**, we can plan on having to tweak up the code in VGtoMF to handle a new set of idiosyncrasies. And, even though I am attempting to write VGtoMF in simple straightforward C code, it will take a bit of work to get it to compile and run whenever we try it on a new system.

Of course, with what we've already said about Generic CADD, the graphic escape sequences for curves and fills would have to work with just colors (they have 256 of them), since CADD does not use "line type" commands. I'm beginning to see that my "graphic escape sequences" are simply a way to implement a set of PostScript graphics commands for a CAD program that does not understand PostScript. Since generic CADD *does* understand PostScript, it would make sense to read the PostScript file directly. It's not much of a design change to enhance the VGtoMF design to handle a small set of PostScript commands, specified (along with the vector graphics commands) in the configuration file.

What should this PostScript command set include?

I've discussed this at some length with Scott van der Linden, who handles the technical support part of the Generic CADD bulletin board on BIX; I've also studied PostScript output from several other commercial CAD programs. Here are the PostScript features supported by Generic CADD:

- Arcs
- Circles
- Bezier curves (4 points per segment)
- Lines
- Ellipses
- Fills (not supported by Macintosh CADD Level1)
- Conversion of colors to greyscales (not supported by Macintosh CADD Level1)

This is pretty close to the list supported by Gold Disk's Professional DRAW on the Amiga. (Professional DRAW also allows the user to import bit-mapped drawings, but **METAPLOT** will ignore them.)

What has to be added to the list of META commands to support this list? Not much! It looks to me like all we need is to add an "ellipse" command and a "fill" command, and generalize the line type command a little bit.

**Ellipse.** Suggested ellipse command:

```
e1(r1,r2,theta);
```

where  $r1$  is the length of the semi-major axis,  $r2$  is the length of the semi-minor axis, and  $theta$  is the angle of the semi-major axis measured counter-clockwise from the positive  $x$  axis.

**Line type.** The syntax of the HPGL line type command is:

```
LT pat_no[,pat_len]
```

where  $pat\_no$  (0..6) specifies a dot/dash pattern and  $pat\_len$  specifies a scale factor for the dashes.

A pattern number 7 (line erase) would be handy. I could add an eighth pattern number indicating Bezier curves, but I'd rather not—this would preclude the possibility of specifying a smooth dashed line. Instead, let's add a third parameter "path\_spec", which is 0 for straight joins (the default) and 1 for Bezier curves (4 points per segment). This permits future extensions like path type 2 (free join), 3 (bounded join), and 4 (tense join). I like it! Here's how the lt command in META language looks after the change:

```
lt(pat_no,pat_len,path_spec);
```

**Fill types.** Earlier we rejected the HPGL "FT" command as merely part of a special-purpose pie-chart and bar-graph complex. Let's resurrect it and look at it:

```
FT [type[,spacing[,angle]]]
```

where "type" can be 1 (solid; bidirectional), 2 (solid; unidirectional), 3 (parallel lines), 4 (cross-hatch), or 5 (ignored).

To include black and white PostScript fills, we need to add type 6 (gray scale) and a fourth parameter (percent) to specify the percent black, so the META command looks like this:

```
ft(type,spacing,angle,percent);
```

"Area erase" is "draw" with fill-type 6 and 0% fill; solid fill could be "draw" with fill type 6 and 100% fill, or possibly just fill type 1 or 2. The PostScript files I've looked at implement "filldraw" by breaking it down into separate fill and draw commands: this lets you draw a solid outline with a dot fill.

More generally, it would be nice to draw a sample and say "Fill area with this pattern."

**Text.** This part is deliberately left vague. HPGL does text by specifying direction (DI) and size (SI) in separate commands and then issuing a label (LB) command. It will take some juggling to define a syntax incorporating METAPLOT's typesetting boxes, HPGL's stick letters, and PostScript typesetting commands (if any—the entry-level CAD programs do not seem to do PostScript typesetting.)

**META commands for PostScript Graphics.** Leaving text for a future article, here is the list of META commands augmented with the tools for doing PostScript graphics:

```
Pen motion:
  pu(x,y);          %move
  pd(x,y);          %draw
  ci(r);            %draw circle
  aa(x,y,theta);    %draw arc
```

```
el(r1,r2,theta);   %draw ellipse
Line specification:
  lt(pat_no,pat_len,path_spec);
  sp(color);
Fill specification:
  ft(type,spacing,angle,percent);
```

**Standard Graphics subsets for PostScript.** I would like to propose a nomenclature for talking about PostScript graphics. I steered clear of PostScript for months because the choices seemed to be either 1) no PostScript or 2) writing a full-featured PostScript interpreter.

In fact, METAPLOT includes a well-defined set of PostScript functions, even though it does not call them PostScript. Let's have some formally-defined subsets of PostScript for Graphics!

METAPLOT is capable of doing "Subset 1 PostScript", which consists of the set of functions supported by Generic CADD Level1 for the Macintosh. (Arcs, circles, Bezier curves, lines, and ellipses in black and white only; fills not supported.)

The META language defined above will support a "Subset 2 PostScript"—same as Subset 1, but add gray scale fill capabilities.

To make it support "Subset 3 PostScript" which has the option of color as well as gray scale fills, we may need to add a "color" parameter to ft; in other words, fill color and outline color for a filled area are typically two different things, and specifying line color does not affect fill color. "Subset 3 PostScript" corresponds to Generic CADD's IBM Level3 product, and also, I believe to Gold Draw's Professional DRAW program for the Amiga.

METAPLOT will probably support "Subset 2" PostScript eventually, but there are no plans to support "Subset 3" PostScript (colored fills).

## Future Directions

The chief items on the menu are

- 1) Formalized METAPLOT support for reading and writing "Subset 1 PostScript". We should soon be able to translate any vector graphics file to PostScript, using an enhanced VGtoMF with the appropriate configuration tables.
- 2) Making METAPLOT work for new users and new CAD programs on new systems. It may take some work to get VGtoMF to compile with non-Amiga C compilers; the METAFONT and T<sub>E</sub>X macros have so far run perfectly on every system we've tried.

On page 306 of this issue of TUGboat is a METAPLOT order form. Be sure to specify diskette size and format in your order! I've tried to set

the price low enough that it won't be a barrier for any of you who wish to join this adventure into the unknown. I look forward to a challenging group effort to see just how many systems we can get **METAPLOT** to work on; and I'm excited about the prospect of illustrations bursting into bloom in  $\text{T}_{\text{E}}\text{X}$  documents all over the world. I'll try to keep the TUGboat readership up to date on future developments.

### Afterward

As we go to press, I've just received my copies of the ANSI *Graphical Kernel System* and *Computer Graphics Metafile* standards. Look at the foregoing paper as a historical treatise on "How Pat Wilcox attained enlightenment on the reasoning behind the inner workings of the Computer Graphics Metafile standard." The standard defines a set of graphical objects very similar to my HPGL-derived list. Two changes are needed: add "elliptical arcs" to my list, and add support for Bezier cubic splines to the CGM standard (splines would be supported as Generalized Drawing Primitives). Add to "Future Directions": incorporate CGM support into the VGtoMF program.

### References

1. Encarnação, J., R. Schuster, and E. Vöge, eds., *Product Data Interfaces in CAD/CAM Applications: Design, Implementation and Experiences*. Springer-Verlag, Berlin Heidelberg New York Tokyo, 1986.
2. Childs, Bart, Alan Stolleis, and Don Berryman, "A Portable Graphics Inclusion." TUGboat, Vol. 10, No. 1, pp. 44-46, April, 1989.
3. Johnson, Nelson, *AutoCAD: The Complete Reference*. Osborne McGraw-Hill, Berkeley, CA, 1989.
4. Harrison, Michael A., "News from the VORTEX Project." TUGboat, Vol. 10, No. 1, pp. 11-14, April, 1989.
5. Rogers, David F., "Computer Graphics and  $\text{T}_{\text{E}}\text{X}$  — A Challenge." TUGboat, Vol. 10, No. 1, pp. 39-44, April, 1989.
6. Kinch, Richard J. "TurboMETAFONT: A New Port in C for UNIX and MS-DOS." TUGboat, Vol. 10, No. 1, pp. 23-24, April, 1989.
7. Tobin, Georgia K. M., *The Elements of METAFONT Style*. Preliminary Version, 4 August 1985.

### Acknowledgments

Tomas Rokicki gets a large part of the credit for **METAPLOT** — first, for his outstanding implementation of  $\text{T}_{\text{E}}\text{X}$  and METAFONT on the Amiga, and second, for being a constant source of inspiration, information, bug fixes, and reassurance as I pushed his software to its outer limits and beyond.

Thanks to my office neighbors at OCLC, Georgia K.M. Tobin and Rick Tobin, for teaching me about  $\text{T}_{\text{E}}\text{X}$  and METAFONT, by a very successful policy of benign neglect coupled with coming instantly to the rescue when I got in trouble. Georgia's instruction manual *The Elements of Metafont Style* [7] was the start of my addiction to METAFONT.

Many thanks also to all the friends and acquaintances who have cheerfully helped out when descended upon by an apparition bearing computer diskettes — "Here, let's see if this will run on your system. Show me your instruction manuals. Can I watch all your graphics programs run? Now can you dump the data files for me? Send me some PostScript!" Some of these long-suffering helpers are (again) the Tobins, who first tried **METAPLOT** with Personal  $\text{T}_{\text{E}}\text{X}$  and showed me MacDRAW II; Bill Hawes (the famed wizard of ARexx on the Amiga); Tim Mooney (the author of mCAD and IntroCAD for the Amiga); Dave Haas of Dartmouth's Northstar Project, who ran **METAPLOT** for me on the Unix system at Dartmouth and is gearing up to be the number one Atari ST beta test site; and Andrea Ardito and Jack Somerville at Foremost Computer Systems, Inc., who opened my eyes to a whole world of Macintosh wonders in a lightning late-night office tour, and dumped their VersaCAD plot files for me. Thanks to Willy Langeveld who sent me PostScript files from VLT, and, most recently, to Scott van der Linden, who has answered a steady stream of questions about Generic CADD for the IBM PC and Macintosh, and has convinced me that they are Doing Things Right.

And, of course, I need to thank John Wilcox, who is putting up with all this nonsense when I really should be working on his program documentation.

◇ Patricia P. Wilcox  
The Coolspring Banjo Works  
6617 Home Road  
Delaware, Ohio 43015

# Output Devices

## TeX Output Devices

Don Hosek

Ordinarily this column includes an exhaustive listing of names and addresses for sources of output device drivers, along with several pages of charts showing what is available. The installment in this issue is much shorter, as you see. There are two reasons for this: the information contained in the charts and the list of sources is being installed in a database and is temporarily in a relatively unprocessable state, and also, there has been no news worth mentioning since the last issue appeared.

Let me therefore take this opportunity to describe the proposed new policy for this column. Effective immediately, the complete list and charts will appear in only issue number 1 of the volume year. The remaining issues will contain only updates. Your comments on this proposal are invited; please send them to both the author and the Editor, whose addresses can be found in the address list starting on page 145.

---

### Report from the DVI Driver Standards Committee

Tom Reid  
Don Hosek

The first few months of 1989 have shown a healthy increase in the DVI driver standards discussion. For those people with network access, much has been done to provide for the dissemination of the information which has come through our hands.

The group has a LISTSERV discussion group, DRIV-L, which is the primary means of communication between its members. The list is set up so that anyone who wants to contribute ideas may do so by sending mail to DRIV-L@TAMVM1 (Bitnet) or DRIV-L@TAMVM1.TAMU.EDU (Internet). These notes will be automatically distributed to the membership of the group.

Archives of past discussions as well as papers on the topic and the current versions of standards documentation, programs, and macros are

stored on the Clarkson archive in the dvi-standard group. Individuals with FTP access may obtain the files from sun.soe.clarkson.edu in the directory pub/dvi-standard. Those without FTP access may still obtain the files via e-mail using the same mechanism as is used by the L<sup>A</sup>T<sub>E</sub>X style collection, substituting dvi-standard for latex-style where appropriate. For example, to obtain the file driv-1.log8809 and a list of other files, one might send a message to archive-server@sun.soe.clarkson.edu which looks like:

```
path fschwartz/hmcvax.bitnet@clvm.clarkson.edu
get dvi-standard driv-1.log8809
index dvi-standard
```

By the TUG meeting in August, we hope to have much of the proposed standard documented and available from the archive.

Bitnet users may also obtain log files from Listserv@tamvm1 by sending the command

```
get driv-1 log yy mm
```

to Listserv@tamvm1 where *yy* is the last two digits of the year and *mm* is the month, expressed as a two digit number. For example, to obtain the log from September, 1988, one would send the command `get driv-1 log8809` to Listserv. Listserv commands should be sent either as the first line of a single-line mail message or as an interactive message (TELL on CMS, SEND on VMS).

For those without network access, the files may be obtained on a floppy disk from John Radel for his usual fees (see the article, "Free" T<sub>E</sub>X software for IBM PCs," page 202, in this issue for information on obtaining these files).

The remainder of this article outlines some preliminary results of the committee's work. Persons interested in implementing portions of this standard should check the Clarkson archive or contact Robert M<sup>c</sup>Gaffey, address on page 145, to obtain the most recent information on the standard.

### 5 \special commands

The committee has decided that the \special commands defined to date will be labeled as "experimental" and later classified as "production" after they've undergone sufficient testing to justify the reclassification. Experimental \special commands are distinguished by the prefix X\_.

Further work on the precise syntactical rules for \special are under development.

#### 5.1 Interface

One of the early decisions of the committee was that \special will be treated as a primitive command

which the end user should never need to type. Instead, `\special` should be accessed through a high level macro set. This has the additional advantage that users at beta test sites will usually not be affected by changes to the syntax or names of `\special` commands. This is important since when a `\special` changes status from “experimental” to “production”, its name will change as noted above.

The committee is developing macros for both plain `TeX` and `LATeX` to interface with the developing standard. At the present time, only preliminary versions of these macros have been written, but a full macro set for both plain `TeX` and `LATeX` should be available by the publication time of this article.

## 5.2 Scope

`\special` commands have been broken down into six classes depending on what portion of the `DVI` output they would affect.

**Global** These `\special` commands affect the entire document. Examples of this class of `\special` include commands for selecting duplex printing or setting the printing orientation (portrait, landscape, etc.).

**Page** These `\special` commands affect only the page on which they are printed. Examples of this class include requests for feeding of special paper from an auxiliary tray (e.g., for a cover sheet) or a single-page change in orientation.

**Box** These `\special` commands affect a block of output that is enclosed in a `TeX` box (and thus is, by necessity, on a single page). For example, a command to rotate a block of text would fall under this class.

**Delimited** These `\special` commands are those that affect a block of output which is not necessarily enclosed by a `TeX` box or contained entirely on a single page. For example, a `\special` command to set color would fall into this class.

**Output generating** These `\special` commands are those which generate self-contained output of some sort. For example, the `Xvec \special` of Section 5.3 falls into this class.

**Attribute setting** These `\special` commands modify the next output generating command which appears on the current page. If no output generating command follows an output modifying command, the command is ignored and the `DVI` driver program should issue a warning. An example of this class of commands would be the `Xlinewidth \special` described in Section 5.3.

The remainder of this section will consist of additional notes on those classes of `\special` commands which need additional comment.

## Global specials

Global specials, it has been decided, will be required to appear on the first page of the document. They will either be identified with a prefix (`Xglobal:`), delimited by a pair of `\special` commands (`Xbegin_globals ... Xend_globals`) or some similar scheme.

One issue that has not been decided is whether the first page containing the global `\special` commands should be the first page of text or a special page on its own. Having global options specified as part of the actual first page of text minimizes the impact on existing drivers. However, it does present some problems with existing macro packages in regard to ensuring that the options are output at the right place. This problem stems from the fact that the `\special` commands used to convey the options to the drivers are normally placed in the body of the document. Macro packages which place headline text or entirely separate title pages prior to writing the first part of the “body” of the document will cause text to appear in the `DVI` file before the global options. Headline text may or may not have any impact upon the global options, but separate title pages will prevent the global options from being on the first page of the `DVI` file. To get around this problem, the mechanism used for passing global information will need to “cooperate” with the output routine within the macro package.

Requiring an entirely separate page at the start of the `DVI` file avoids the need for special interaction with the output routines of various macro packages. Instead of placing `\special` commands in the body of the first page, a separate macro is used which issues a separate `\shipout` containing the `\special` commands. This approach makes things easier for programs which sort or otherwise reorganize a `DVI` file since no culling of global options from the first text page is necessary. However, the separate page technique has an undesired effect: it produces a blank page on existing drivers which do not understand the options page.

## Box specials

A box `\special` command, since it will always be entirely typeset on a single page, will be enclosed in a `TeX` box (`\hbox` or `\vbox`). In the `DVI` output, box structure is reflected by surrounding `push` and `pop` commands. For example, the `TeX` commands:

```
normal
```

```
\hbox{\special{abc} special}
text
```

generate the following DVI code:

```
"normal"
push
right
xxx "abc"
"special"
pop
right
"text"
```

A DVI driver can exploit this for a command such as `X_rotate` by maintaining on the DVI stack, values for items such as `rotation_angle`.

### Delimited specials

The committee has not found an effective way to deal with open block `\special` commands yet. They will probably need to be issued in cooperation with the output routine, to insure that every delimited command is broken down into matching pairs of `\special` commands on each page within its bounds.

This approach is necessary for two reasons:

- If pages are reordered for any reason (*e.g.*, reverse ordering for laser printers which stack output face up) the driver should not need to have to scan the entire file to insure that it does not inadvertently break up a pair of `\special` commands producing a delimited command.
- Without special care being taken, a delimited command which spans pages may inadvertently affect page headers and footers which are typeset between the beginning and ending blocks.

### 5.3 Graphics commands

Three techniques for including graphics have been discussed. These are:

1. Make graphics entirely with `TeX` primitives.
2. Use `METAFONT` to build a graphic as a font.
3. Allow the driver to include a device-specific graphic.

#### Graphics by `TeX`

Handling graphics entirely with `TeX` macros and primitives which use dots or characters from a special graphics font is a technique which has been in use for some time. The `LATeX` picture environment and `PTCTEX` work in this way with the former assembling characters from a graphic font and the latter using closely spaced dots.

In TUGboat 10(1),<sup>1</sup> David F. Rogers proposed a series of `TeX` macros to provide plotting primitives; these macros would generally be used by `TeX` input generated by some graphics package. The macros which were proposed created graphics by closely spacing dots along each line in the same manner as `PTCTEX`.

The problem posed by creating graphics in this manner is that `TeX` must store all of the graphic elements in memory at once for an entire page, possibly exceeding `TeX`'s capacity.

To calculate the memory needs, the technique for positioning each dot was specified as:

```
\kern\DX \raise\Y \hbox{\DOT}}%
```

where `\DX` is a dimension register giving the displacement in the "x" direction from the previous point and `\Y` is a dimension register giving the displacement in the "y" direction from the reference point of the graph. `\DOT` defines the plotting symbol and `\DX` accounts for the width of this symbol.

In memory, `TeX` saves `\kern\DX` in a *kern* node, the raised `hbox` in an *hlist* node, and the plotting symbol in a *char\_node*. These take two words, seven words, and one word of memory, respectively, for a total of ten words per dot. A normal-size implementation of `TeX` with 64 k-words of memory allows about 6000 dots to be positioned before it runs out of memory (assuming that no other macros are loaded and neglecting other text on the page). Spacing the dots at 100 per inch, this gives about 60 inches, which is not sufficient for many graphs.

To enhance the capacity of this graphics technique, we decided to use a `\special` to add a vector drawing capability to `TeX` and DVI drivers and use the `\special` instead of closely-spaced dots. This changes the `TeX` command sequence to:

```
\kern\DX \raise\Y
\hbox{\special{X_vec \number\XC
\space \number\YC}}%
```

where `\XC` and `\YC` are dimension registers giving the components of the vector. Component values in scaled points are likely to be six-digit numbers with an additional minus sign for negative numbers. Thus, an average length for the `\special` string is likely to be around 18 characters. In memory, a `\special` is saved in a two-word *whatsit* node which points to the `\special` string. Thus the total memory needs, counting the *kern* and *hlist* nodes, will average 29 words per vector which allows roughly 2000 vectors. This may be sufficient for many graphs, but falls somewhat short for complex three-dimensional

<sup>1</sup> This article also appeared in `TeXhax` 89(7).



surface plots. (One sample 3D surface plot consisted of 13,000 vectors.)

Two `\special` commands have been defined for graphics of this sort (and specialized commands for more complicated graphic elements will be defined in the future). The commands defined are:

`X_linewidth n` Specify that the following vector is to be drawn with a line width of *n* DVI units (scaled points for `TeX`). Vectors are normally 1 point in width. If no vector follows the `X_linewidth \special` on this page, the command is ignored and the DVI driver program should issue a warning.

`X_vec  $\Delta x$   $\Delta y$`  Draw a diagonal line from the current point to the point which is offset by  $\Delta x$  and  $\Delta y$  from the current point.  $\Delta x$  and  $\Delta y$  are specified in terms of DVI units.

### Graphics by METAFONT

A different approach to graphics inclusion is to use METAFONT to produce the graphic as a character of a font and position it using `TeX`'s normal character positioning capabilities. The advantage of this technique is that the graphic is in a format which many drivers will already accept.

METAPLOT by Pat Wilcox<sup>2</sup> is one example of a package which takes this approach.

However, the technique has a number of drawbacks: Graphic fonts are resolution-dependent; a separate graphic font is needed for different resolution devices. METAFONT records changes in pixel values across a scan line when it builds a character. Thus, the memory needs depend upon the complexity of the graphic in addition to the size and resolution of the device. To circumvent this limitation, it is necessary to break the whole graphic into smaller pieces. It is important to ensure that the heights and widths of each piece are integral numbers of pixels to allow them to be reassembled without the alignment problems which occur for letters within words.

### Including device-dependent files

With this approach, the DVI driver processes a special Graphics Description File (GDF) which, among other things, indicates the names and formats of separate graphic files in device-dependent format. A driver searches this list to find a file in a format appropriate for the device it supports. This allows a greatly simplified graphic files to be defined for pre-

viewing purposes while a detailed, higher resolution version is used when the DVI file is printed.

GDF files are processed both by `TeX` and by the DVI driver. `TeX` `\input`s the file and executes code at the start of the file. This code sets some dimension and box registers giving the size of the graphic then terminates with an `\endinput` to return control to the macro which did the `\input`. The portion of the GDF file following the `\endinput` is processed by the driver.

The driver section of the file consists of a series of keywords which identify lines that apply to a particular graphics format, rotation, etc. The driver scans these lines searching for a format which it understands. Depending on the driver and the graphics format, additional lines may have to be searched for other attributes such as rotation. Eventually, the name of the graphics file to be included will be found and the driver will incorporate it into the output file.

In "A portable graphics inclusion" (TUGboat 10(1)), Bart Childs, Alan Stolleis, and Don Berryman suggested another scheme for using `\special` to include device-dependent graphics files.

### 6 Additional reference material

In addition to the works mentioned in the Editor's note at the end of our last report, the following may also be of interest:

- Guntermann, Klaus and Joachim Schrod. "High quality DVI drivers". Available from the Clarkson archive as the file `schrod-guntermann1.tex`.
- Hosek, Don. "Proposed DVI `\special` command standard". Available from the Clarkson archive as the file `hosek1.tex`.

In addition, anyone interested in implementing any portion of the developing standard should read the logs available from the Clarkson archive or `Listserv@tamvm1`.

◊ Tom Reid  
Computing Services Center  
Texas A&M University  
College Station, TX 77843  
Bitnet: `X066TR@TAMVM1`

◊ Don Hosek  
3916 Elmwood  
Stickney, IL 60402  
Internet: `u33297@uicvm.uic.edu`  
Bitnet: `u33297@uicvm`  
Bitnet: `dhosek@ymir`  
UUNet: `dhosek@`  
`jarthur.claremont.edu`  
JANET: `u33297%uicvm.uic.edu@`  
`uk.ac.earn-relay`

<sup>2</sup> See page 179 of this issue of TUGboat for information about this package; see also the Amiga`TeX` notes of March 12, 1989 or `TEXMAG` 3(3).

## Resources

### Announcing (belatedly) $\text{T}_{\text{E}}\text{X}_{\text{M}}\text{A}_{\text{G}}$

Don Hosek  
University of Illinois at Chicago

After being chided for not publicizing my electronic "magazine" enough, I have decided to make a formal announcement of its availability to the  $\text{T}_{\text{E}}\text{X}$  community at large.

### What is $\text{T}_{\text{E}}\text{X}_{\text{M}}\text{A}_{\text{G}}$ ?

$\text{T}_{\text{E}}\text{X}_{\text{M}}\text{A}_{\text{G}}$  is available free of charge to anyone reachable by electronic mail and is published approximately every two months. The subject material generally falls somewhere between the somewhat chaotic (but still useful) correspondence of  $\text{T}_{\text{E}}\text{X}_{\text{H}}\text{A}_{\text{X}}$  and  $\text{U}_{\text{K}}\text{T}_{\text{E}}\text{X}$ , and the printed matter in *TUGboat* and *T\_{\text{E}}\text{X}line*. Some previous articles have included an early version of Dominik Wujastyk's article on fonts from *TUB 9#2*; an overview of the different font files used by  $\text{T}_{\text{E}}\text{X}$ ,  $\text{M}_{\text{E}}\text{T}_{\text{A}}\text{F}_{\text{O}}\text{N}_{\text{T}}$ , and device drivers; macros for commutative diagrams and simple chemical equations and many other topics. One issue was dedicated to the issue of non-English  $\text{T}_{\text{E}}\text{X}$ .

### How do I subscribe?

You can only subscribe if you have access to one of the electronic mail networks and can send mail to Bitnet (I have neither the time nor resources to mail hardcopy issues to those without network access). To subscribe, one should send the following one line message to `listserv@pucc.bitnet` or `listserv@pucc.princeton.edu`:

```
SUBS  $\text{T}_{\text{E}}\text{X}_{\text{M}}\text{A}_{\text{G}}\text{-L}$  your full name
```

If you have problems doing this, send a note to `U33297@uicvm.uic.edu` asking to be added to the list (this address sends mail to me, *not* a server, so phrase it for human reading).

There are also several "regional" redistributions. CDNnet subscribers may subscribe by sending a note asking for a subscription to  $\text{T}_{\text{E}}\text{X}_{\text{M}}\text{A}_{\text{G}}$  to `list-request@ucb.csnet`. Janet subscribers should request subscriptions from Peter Abbott, `Abbott@Uk.Ac.Aston`.

### Where can I get back issues?

Users with FTP access to the internet may retrieve back issues of  $\text{T}_{\text{E}}\text{X}_{\text{M}}\text{A}_{\text{G}}$  from the directory `pub/texmag` on `sun.soe.clarkson.edu`.

Janet users may obtain back issues from the Aston  $\text{T}_{\text{E}}\text{X}$  repository (for details, contact Peter Abbott, e-mail address above). DECnet/SPAN users may obtain back issues from the European (contact Massimo Calvani, `fisica@astrpd.infn.it`) or American (contact Ed Bell, `7388::bell`) DECnet  $\text{T}_{\text{E}}\text{X}$  repositories.

Others with network access should send a message to `archive-server@sun.soe.clarkson.edu` with the first line being `path` followed by an address *from* Clarkson *to* you, and then a line

```
get texmag texmag.v.nn
```

for each back issue desired where *v* is the volume number and *nn* the issue number. The line `index texmag` will give a list of back issues available.

A typical mail request may resemble:

```
index texmag  
get texmag texmag.1.08
```

### How do I submit articles to $\text{T}_{\text{E}}\text{X}_{\text{M}}\text{A}_{\text{G}}$ ?

I was hoping you would ask. Articles are accepted on all aspects of  $\text{T}_{\text{E}}\text{X}$ ,  $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ , and  $\text{M}_{\text{E}}\text{T}_{\text{A}}\text{F}_{\text{O}}\text{N}_{\text{T}}$  from specific information on interfacing graphics packages with particular DVI drivers to general information on macro writing to product reviews to whatever else strikes your fancy. A general rule of thumb to use in deciding whether something would make a suitable  $\text{T}_{\text{E}}\text{X}_{\text{M}}\text{A}_{\text{G}}$  article is to assume that it would!

$\text{T}_{\text{E}}\text{X}_{\text{M}}\text{A}_{\text{G}}$  has two special columns for shorter submissions as well: "The Toolbox" is a forum for presenting short useful macros, and "TeX Mysteries and Puzzles" presents interesting and unusual typesetting problems for possible solutions by the  $\text{T}_{\text{E}}\text{X}_{\text{M}}\text{A}_{\text{G}}$  readership (these questions are also forwarded to the  $\text{T}_{\text{E}}\text{X}_{\text{H}}\text{A}_{\text{X}}$  and  $\text{U}_{\text{K}}\text{T}_{\text{E}}\text{X}$  groups in hopes of getting as many responses as possible). The purpose of both of these columns is to attempt to provide *exposition* of problem solving  $\text{T}_{\text{E}}\text{X}$ niques, so when submitting macros for either of these, please try to explain how and why you did what you did.

◇ Don Hosek  
3916 Elmwood  
Stickney, IL 60402  
Internet: `u33297@uicvm.uic.edu`  
Bitnet: `u33297@uicvm`

---

**TeXline**

Malcolm W Clark

*TeXline* now terms itself 'a newsletter of the TeX community'. When it began in 1985, it described itself as a newsletter for TeX users in the UK and Ireland. The TeX-world has changed even in the last four years.

In 1984, I attended the historic TUG meeting at Stanford where DPK announced that TeX had been finalised, and that no more work was to be done on it, with the exception of bug fixes. At the same meeting I saw TeX demonstrated on the Sun, under a windowing environment, where input text and a preview could be viewed side by side. I returned to the Old World determined to be a missionary. TeX was alive in Britain, but I felt that we had to try to identify one another, and started to build a mailing list from the TUGboat listings (an arduous task, since even to this day there are no regional sub-divisions provided<sup>1</sup>). Encouraged in my foolhardiness by others I decided to start a newsletter, based on that mailing list. From the outset, *TeXline* has been free. Obviously somebody has paid for the raw materials and the postage, but we just don't enquire too deeply about that.

The newsletter format has been fairly consistent, and frankly, pretty mundane. Because of a fundamental restriction to easily obtainable raw materials (A4 paper), I use double columning, with a basic 10 point typeface. I have yet to find a really robust set of double columning macros. The first editions were set using TeX80 (a slightly augmented TeX78) running on a CDC machine under NOS, and with an Autologic APS- $\mu$ 5 phototypesetter for output device. The pages were pasted up from a sort of galley. I also included material prepared by others from a variety of devices (embarrassingly, this also included a typewriter). Since then production has switched first to MicroTeX and a LaserWriter, and now uses *Textures* on a Mac Plus (again with a LaserWriter). The quantity of paste up has diminished to almost nothing (but not for ideological reasons—I still feel that paste up is often the best way to tackle problems).

What goes into *TeXline*? As all editors will realise, what goes into most newsletters is a mixture of what you are prepared to write yourself and what you can extract from your friends. If you sit and

wait for articles to arrive, you sit and wait a long time. I feel that *TeXline* has been well supported, even if I do write far too much of it still. I try not to edit very much, but it is always necessary to re-word (and sometimes re-write) in order to make articles fit into pages. Sorry. The next issue of *TeXline*, number 9, will have a table of contents for all the previous ones. I try to include areas which are not strictly TeXware (where TeXware includes L<sup>A</sup>T<sub>E</sub>X, A<sub>M</sub>S-TeX, METAFONT, etc.), like SGML, POSTSCRIPT matters, reports of relevant meetings (like those of the British Computer Society's Electronic Publishing Group, the SGML Users Group, TUG, and anything similar). Software and book reviews are becoming more frequent. I even managed to persuade one brave soul to review *The TeXbook*. There is a fair amount of very mainstream TeXnical stuff which would not be too out of place in TUGboat. Barbara Beeton and I have discussed the possibility of reprinting some *TeXline* articles in TUGboat. I have recently increased the amount of plagiarism by using material which has appeared over the electronic networks, or which goes into other newsletters.

*TeXline* tries to maintain a degree of informality—at least that's my excuse for the typos. Many of its readers do not have access to electronic networks, and, to my chagrin, the majority are not even TUG members. I believe that it is very important to get out into the community of 'little people'—the ones who do not work in organisations where there are lots of fellow TeXies; with the successful porting of TeX to personal machines, the possibilities of isolation have increased. I would like to think that *TeXline* was addressing that situation (without ignoring the other parts of the community).

As the newsletter grew (in bulk), its mailing list also grew, especially outside the UK and Ireland. This was the main reason for changing the 'subtitle' so that the newsletter merely described itself in more general, non-nationalistic terms. This however represents a problem. Mailing costs for the minority outside the British Isles now dominate the costs. I try to arrange re-distribution centres. TUG has agreed to provide a subsidy. Some of the small surplus from last year's TeX88 conference has also helped to offset costs.

What next? I once said that the newsletter would never be electronic. Without exactly eating my words, future editions (and some past ones) will be included in the Aston TeX Archive. This may mean that the newsletter can be printed elsewhere (although there will be problems where graphics are included). It is not intended to remove the need for printed paper copies. *TeXline* is set

---

<sup>1</sup> Editor's note: Clearly Malcolm had not seen this year's membership list when he wrote this. A listing by country and city has been added, by popular demand.

using  $\text{\TeX}$  — the layout may not be exciting, but it is part of the newsletter. I would like to see it come out more regularly (even if not more frequently). And I'd like to see more people volunteer articles. You can submit articles electronically — to `texline@vaxa.cc.imperial.ac.uk`

if you have JANET access

`mwc@doc.ac.uk`

if you have UNIX/UUCP access

But if you must send your request to be added to the mailing list by this means (together with your article of course), please, please, pretty please, give me your street address. Remember that *TeXline* is produced on paper, and that paper is not so easily transmitted electronically (yet). I really don't like typing stuff in myself. It is awfully boring and error-prone. Floppy disks travel remarkably well — that's a hint. Send them to:

Malcolm Clark  
Imperial College Computer Centre  
Exhibition Road  
London SW7 2BP  
UK

That way, you are guaranteed a place on the mailing list.

---

## UK $\text{\TeX}$ and the Aston Archive

Peter Abbott  
Aston University UK

At the time of writing this report (May 1989) the reorganisation of the archive is in full swing. Since UK $\text{\TeX}$  is now available in a number of archives it is safer to say read those for the most up-to-date information.

The archivists, listed in TUGboat Vol. 10 no. 1, have been extremely active and there now exists considerable supporting documentation which allows users to navigate there way around the archive as well as giving details of the required elements for building a version of  $\text{\TeX}$  for the target system. MAC users are now catered for in that Oz $\text{\TeX}$  (described on page 202) is available in HQX format which is mailable from the archive. MS DOS systems have likewise been catered for with mailable versions of the relevant PC software.

The problem of `stream_1f` files has, hopefully, been overcome and access to **ALL** items in the archive via mail should now be possible.

There are still two major impediments to using mail for accessing the archive:

- Character tables at Gateways
- Limits on the size of mail messages.

The mail server will be amended to enable large files to be sent in smaller units to avoid the second problem but as yet there is no satisfactory way of eliminating the character translation problem.

Access from JANET sites to the archive is relatively simple and painless. The FTP facilities that are provided can be illustrated by a simple example. (I shall use the VAX/VMS notation, but there are corresponding formats for UNIX, VM/CMS, NOS/VE etc). A user called `orinocco` is registered on a system with the name `uk.ac.wimbledon.common`. To extract files from the archive, `orinocco` signs on to his system and types the command `transfer`. The required parameters are input filename, output filename, remote username, remote username, password. If we assume `orinocco` wishes to fetch the file `[tex-archive]00readme.txt`, the sequence is

```
transfer
%_Input filename?
uk.ac.aston.tex::[tex-archive]00readme.txt
%_Output filename? archive.list
%_Remote username? public
%_Remote username password? public
```

Transfer nnnnn has been queued.

Sometime later the file is available on the system at `uk.ac.wimbledon.common`. Failure to find or transfer the file and other error messages are notified via the normal VAX/VMS mail system.

The Aston mail server is a batch job which runs on a VAX 8650 processor under the VAX/VMS operating system. Sometimes genuine mail disappears for one reason or another, so if no reply is received after a suitable period you are recommended to try again. It is impossible to give estimates of the turnaround time for any individual user; the server runs once per hour and the mail messages are queued for transmission. The mail software makes a maximum of 30 attempts to send a message (10 at 10 minute intervals, 10 at 1 hourly intervals and 10 at 4 hourly intervals). This rather extended period is designed to overcome short-term network failures and for systems which are switched off for short periods of time or overnight. The cluster system at Aston is normally available 24 hours a day, seven days a week with the occasional booked systems maintenance on

a Wednesday morning and twice yearly maintenance checks by DEC.

Instructions on how to extract files from the archive are contained in a help file, and this file is available by sending a mail message to

`texserver@uk.ac.aston.tex`

UK addresses on JANET are big-endian format and most users 'on the other side of a gateway' will need to specify it as `texserver@tex.aston.ac.uk`. The subject line in the incoming mail message is ignored, as is any text, until a line starting with --- (three minus or hyphen characters in columns 1 to 3); any text on that line is also ignored. The next line is the `name@return` address in UK format and the third line is the word `help` (in UPPER, lower or MiXeD case). For example:

```
--- (any text on this line is ignored)
name@address
help
```

The best rule to observe in quoting `name@address` is to use the format:

- JANET sites  
`name@uk.ac.site.system`
- Sites via earn-relay (Internet, Earn)  
`name%little-endian%big-endian@earn-relay`
- Internet sites may be able to use  
`name%little-endian%big-endian@nsfnet-relay`
- Sites via uk.ac.ukc (UUCP)  
`name%little-endian%big-endian@uk.ac.ukc`

Anyone who has problems getting mail back is welcome to send me (`abbott@uk.ac.aston`) the message that they have tried and I will forward it to

`texserver@uk.ac.aston.tex`

with a copy to the originator showing the `name@address` format that is required. I do not guarantee to be able to solve every query but will do my best.

◊ Peter Abbott  
Computing Services  
Aston University  
Aston Triangle  
Birmingham B4 7ET, England  
Internet: `pabbott@nss.cs.ucl.ac.uk`

---

## The DECUS TeX Collection

M. Edward Nieland

### 7 Introduction

The DECUS TeX Collection is a collection of freely distributable files of TeX and TeXware for computers found in DEC (Digital Equipment Corporation) sites. The collection is made available by the DECUS Language and Tools SIG (Special Interest Group) Public Domain Working Group. It is available from DECUS Library and through DECUS LUGs (Local User Groups).

The collection currently covers three operating systems: VMS, UNIX, and MS-DOS. The collection is distributed in VMS BACKUP format (one 6250 BPI tape, one TK-50, or two 1600 BPI tapes). The collection contains a ready-to-run set up for the VMS operating system.

The collection includes executables, fonts, style files, and source. The version date of the current collection is February 1989.

### 8 What is available?

The DECUS TeX collection is one of the largest collections of TeXware available in the United States. It contains over 54 megabytes of TeX material.

Included are:

- TeX 2.95
- LaTeX 2.09
- SlitEX 2.09
- METAFONT 1.7
- TeXsis 2.11.5
- BibTeX .99c
- AMS-TeX
- PICTEX
- WEB
- METAFONT Tools (GFtoPK, PXLtoPK, GFTODVI, etc.)
- LaTeX Style Collection
- Utah DVI Driver Collection
- GLOTeX
- IDXTEx
- TeXTYL
- DVItO LN03
- LN03DVI
- DVI2PS
- DVITOV DU 3.0
- PSPRINT 3.0
- Adrian Clark's Edit interface to VMS TeX
- Fonts designed for a LN03

- Halftone Fonts
- SPELL (VMS Spelling checker that understands  $\TeX$  and  $\LaTeX$ )
- DVIDIS (Previewer for VAXStations)
- RNOto $\TeX$
- SCREENVIEW
- PS $\LaTeX$
- Templates for LSE
- PSFIG
- TGRIND
- TR2 $\TeX$

The MS-DOS material is included in ARC files:

- DOST $\TeX$
- SB $\TeX$
- CDVI (previewer)
- DVIVGA (previewer)
- DVIEW (previewer)
- Fonts

The following UNIX material is included in compressed TAR files:

- WEB2C
- COMMON $\TeX$
- S2 $\LaTeX$
- PIC2FIG
- PSFIG
- MFWARE
- IP $\TeX$
- DE $\TeX$
- FIG-FS
- Bib $\TeX$ -IN-C
- TIB
- TRANSFIG
- $\TeX$ IDX
- MAKEINDEX
- P $\Gamma$ C $\TeX$

The sources to  $\TeX$ x and VXDVI previewers for X11 are also included.

### 9 How do I get a copy?

The DECUS  $\TeX$  Collection can be obtained from the DECUS Library for a minimal charge (cost to cover expense). The order number is V-SP-58. To order contact:

The DECUS Program Library  
219 Boston Post Road BP02  
Marlboro, MA 01752-1850  
Phone: 508 480-3418

The DECUS  $\TeX$ Collection is also available via the DECUS National LUG Organization Tape Copy Project. The Tape is made available to DECUS LUGs at no charge (you provide the tape). Contact your local LUG to see about getting a copy of the tape. If you don't know how to contact your LUG, contact DECUS at (508) 480-3446 to find out.

### 10 Additions to the collection

Additions to the DECUS  $\TeX$  Collection are accepted and encouraged. Submissions and suggestions for submissions can be sent to the collection editor:

M. Edward (Ted) Nieland  
Systems Research Laboratories, Inc.  
2800 Indian Ripple Road  
Dayton, OH 45440-3696  
Internet: tnieland@amr1.af.mil  
Phone: (513) 255-8846

---

### Contents of Archive Server as of 1 May 1989

Michael DeCorte

Due to the size of the archive, from now on the first issue of the year will contain the complete list of files and the following issues will only contain the new and updated files.

As always, submissions are encouraged. If you do submit a file please include at the top of the file: your name; your email address; your real address; the date. Also please make certain that there are no lines in the file longer than 80 characters as some mailers will truncate them. Mail should be sent to  
mrd@sun.soe.clarkson.edu  
archive-management@sun.soe.clarkson.edu

### For Internet users: how to ftp

An example session is shown below. Users should realize that ftp syntax varies from host to host. Your syntax may be different. The syntax presented here is that of Unix ftp. Comments are in parentheses. The exact example is for retrieving files from the  $\LaTeX$  Archive; the syntax is similar for the other archives, only the directories differ. The directory for each archive is given in its description.



### Distribution for IBM PC and clone users

There are two sources.

- David W. Hopper  
446 Main Street  
Toronto, Ontario  
Canada M4C 4Y2

has L<sup>A</sup>T<sub>E</sub>X style files only. David has in been in a state of flux for a little while and would like to apologize for any delays. If you have not received requested files from him you should get in contact with him. You should send:

1. either one 1.44 MB 3.5 inch diskette, one 1.2 MB diskette or four 360 KB diskettes, blank and formatted;
2. indication of the format required;
3. a self-addressed mailer; and
4. a \$5.00 donation per set of files, to cover postage and equipment wear & tear. (If you live outside North America, airmail delivery will probably require more postage. You should probably contact David for details.)
5. No phone calls or personal visits please.

- Jon Radcl  
P. O. Box 2276  
Reston, VA 22090

has L<sup>A</sup>T<sub>E</sub>X style files and other material including T<sub>E</sub>X. For a list of what is available and other information send a SASE.

### A<sub>M</sub>S-T<sub>E</sub>X Sources

This directory contains the T<sub>E</sub>X source needed to build A<sub>M</sub>S-T<sub>E</sub>X, and is a duplicate directory of `tex.amstex` on Score. Files are located in `pub/amstex` for ftp users. Mail users should request files from the `amstex` archive.

### BIBT<sub>E</sub>X Sources

This directory is a duplicate of `tex.bibtex` on Score, and contains the BIBT<sub>E</sub>X style files and the WEB files needed to build BIBT<sub>E</sub>X. Files are located in `pub/bibtex` for ftp users. Mail users should request files from the `bibtex` archive.

### CM Fonts

This directory contains the METAFONT files needed to build the CM fonts, and is a duplicate of `tex.cm` on Score. Files are located in `pub/cm-fonts` for ftp users. Mail users should request files from the `cm-fonts` archive.

### DVI Driver Standards

This directory contains digests from the DVI Driver standards committee. Files are located in `pub/dvi-standard` for ftp users. Mail users should request files from the `dvi-standard` archive. Files are named `driver.YY.MM` where YY is the year of the issue, MM is the month. There are also articles about DVI standards here.

### L<sup>A</sup>T<sub>E</sub>X Sources

This directory is a duplicate of `tex.latex` on Score, and contains the T<sub>E</sub>X files needed to build L<sup>A</sup>T<sub>E</sub>X. Files are located in `pub/lamport` for ftp users. Mail users should request files from the `lamport` archive.

### METAFONT Sources

This directory contains the WEB files needed to build METAFONT. It is a duplicate of `tex.mf` on Score. Files are located in `pub/mf` for ftp users. Mail users should request files from the `mf` archive.

### T<sub>E</sub>X Documentation

This directory contains documentation on T<sub>E</sub>X. It is a duplicate of `tex.doc` on Score. Files are located in `pub/lamport` for ftp users. Mail users should request files from the `lamport` archive.

### T<sub>E</sub>X Inputs

This directory contains the T<sub>E</sub>X files needed to build plain T<sub>E</sub>X. It is a duplicate of `tex.inputs` on Score. Files are located in `pub/tex-inputs` for ftp users. Mail users should request files from the `tex-inputs` archive.

### T<sub>E</sub>X Sources

This directory is a duplicate of `tex.web` on Score, and contains the WEB files needed to build T<sub>E</sub>X. Files are located in `pub/tex-source` for ftp users. Mail users should request files from the `tex-source` archive.

### T<sub>E</sub>X Tests

The directory contains the files needed to test T<sub>E</sub>X using the `triptest`. It is a duplicate directory of `tex.tests` on Score. Files are located in `pub/tex-tests` for ftp users. Mail users should request files from the `tex-tests` archive.

### T<sub>E</sub>Xhax Digests

This directory contains all of the back issues of T<sub>E</sub>Xhax. Files are named `texhax.YY.NNN` where YY is the year of the issue and NNN is the issue number.



Files are located in `pub/texhax` for ftp users. Mail users should request files from the `texhax` archive.

### TEXMAG Digests

This directory contains all of the back issues of TEXMAG. Files are named `texmag.V.NN` where `V` is the volume number and `NN` is the issue number. Files are located in `pub/texmag` for ftp users. Mail users should request files from the `texmag` archive.

### Transfig Collection

This directory contains the C source for Transfig; a program that converts Fig output to other forms such as PICTEX. Files are located in `pub/transfig` for ftp users. Mail users should request files from the `transfig` archive.

### TUGboat Files

This directory contains files related to TUGboat and is a duplicate of `tex.tugboat` on Score. Files are located in `pub/tugboat` for ftp users. Mail users should request files from the `tugboat` archive.

### UKTEX Digests

This directory contains all the back issues of UKTEX. Files are named `uktex.YY.NNN` where `YY` is the year of the issue and `NNN` is the issue number. Files are located in `pub/uktex` for ftp users. Mail users should request files from the `uktex` archive.

### AMS-TEX

This directory contains style files specific to AMS-TEX users. Files are located in `pub/amstex-style` for ftp users. Mail users should request files from the `amstex-style` archive.

`mssymb.sty` the definitions for the symbols in the two "extra symbols" fonts created at the AMS

### BIBTEX

This directory contains files that are specific to version 0.99 of BIBTEX. Many of these files are to be used with files in the LATEX Collection. Files are located in `pub/bibtex-style` for ftp users. Mail users should request files from the `bibtex-style` archive.

`named.bst` for use with `ijcai89.sty`

### BIBTEX 0.98 Collection

This directory contains files that are specific to version 0.98 of BIBTEX. Many of these files are to be used with files in the LATEX Collection.

Files are located in `pub/bibtex-style-0.98` for ftp users. Mail users should request files from the `bibtex-style-0.98` archive.

`btxbst.doc` A master file for BibTeX styles with standard styles and some new ones.

### LATEX Style Files

This directory contains files that are specific to LATEX. Most of these are style files but some of them are programs. Some of the files support BIBTEX style files that are in the BIBTEX Collection or the BIBTEX 0.98 Collection. Files are located in `pub/latex-style` for ftp users. Mail users should request files from the `latex-style` archive.

`agugrl-sample.tex` AGU Geophysical Research Letters style

`agujgr-sample.tex` AGU Journal of Geophysical Research style

`cd.sty` Commutative diagram macros

`cd-doc.tex`

`breakcites.sty` allows citations to break across lines

`bsf.sty` provide access to bold san serif fonts in LATEX

`deproc.readme`

`deproc.sty` DECUS proceedings style and documentation

`eepic11b.shar` a picture environment that used tpic specials

`fancyheadings.sty` modify the headers and footers

`fullpage.sty` get more out of a page

`german.sty` style file for German

`hackalloc.sty` make allocation local for LATEX

`ijcai89.sty` Conference style for IJCAI-89

`ijcai89.tex`

`jeep.sty` useful modifications of the article style

`jeep.tex`

`ltugbot.sty` for articles to tugboat

`mitpress.sty` a simple MIT Press format

`mf.sty` make METAFONT logos at all sizes

`named.sty` for use with `named.bst`

`natsci.sty` natural sciences style (BIBTEX file in `bibtex-style-0.98`)

**pagefoots.sty** puts footnotes at the bottom of each page

**parskip.sty** sets parindent to 0 and puts some glue into parskip to aid page breaks

**portland.sty** environments to switch between portrait mode and landscape mode

**refman.sty** document style for reference manuals similar to the PostScript manual

**res.sty** a format for doing resumes by Michael DeCorte

**resume.sty** a format for doing resumes by Stephen Gildea

**svlncs.sty** a document style for articles in books printed in the Springer-Verlag LNCS series

**verbatimfiles.sty** include a file in a verbatim mode

### TeX Fonts

This directory contains the METAFONT files for user contributed fonts. Files are located in pub/tex-fonts for ftp users. Mail users should request files from the tex-fonts archive.

**apl.shar** APL fonts and related macros

**tengwar.shar** the fonts used by Tolkien in Lord of the Rings

**wujastyk.tzh** description of a lot of different fonts

**greek1.shar** for papers in Greek

**greek2.shar**

**hershey.pas** Hershey fonts

**hershey.tzh**

**acwtosc.pas**

**hershey-test.tex**

**orient.mf**

**xhershey.shar** Hershey script fonts and a program to convert vector fonts to METAFONT

**ccr5.mf** the fonts and macros for **Concrete Mathematics**

**ccr6.mf**

**ccr7.mf**

**ccr8.mf**

**ccr9.mf**

**ccr10.mf**

**cccsc10.mf**

**ccmi10.mf**

**ccsl10.mf**

**ccti10.mf**

**ccslc9.mf**

**gkpmac.tex**

**ocr-a.mf** OCR-A fonts by Tor Lillqvist

**ocr-ai.mf**

**ocr-aii.mf**

**ocr-aiii.mf**

**ocr-aiv.mf**

**cmpica.mf** CM Pica by Don Hosek

**cmpicab.mf**

**cmpicati.mf**

**pica.mf**

**pcpunct.mf**

**b-circle.mf** John Sauter's reparameterized Computer Modern. To create an arbitrary CM font is to create a file with the following two lines:  
**design\_size:=SIZE;**  
**input b-FONT**  
 This will produce the typeface FONT with design size SIZE. for example, if FONT is cmr and SIZE is 11, you will get cmr11.

**b-cmb.mf**

**b-cmbsty.mf**

**b-cmbx.mf**

**b-cmbxsl.mf**

**b-cmbxti.mf**

**b-cmcsc.mf**

**b-cmdunh.mf**

**b-cmex.mf**

**b-cmff.mf**

**b-cmfi.mf**

**b-cmfib.mf**

**b-cminch.mf**

**b-cmitt.mf**

**b-cmmi.mf**

**b-cmmib.mf**

**b-cmr.mf**

**b-cmsl.mf**

**b-cmslitt.mf**

**b-cmss.mf**

**b-cmssbx.mf**

**b-cmssdc.mf**

**b-cmssi.mf**

**b-cmssq.mf**

**b-cmssqi.mf**

**b-cmsy.mf**

**b-cmtcsc.mf**

**b-cmtex.mf**

**b-cmti.mf**

**b-cmtt.mf**

**b-cmu.mf**

**b-cmvtt.mf**

**b-lasy.mf**

**b-lasyb.mf**

**b-line.mf**

**b-linew.mf**

**c-circle.mf**

**c-cmbx.mf**

**c-cmff.mf**

**c-cmmi.mf**

**c-cmr.mf**

c-cmss.mf  
 c-cmssbx.mf  
 c-cmssq.mf  
 c-cmsy.mf  
 c-cmti.mf  
 c-cmtt.mf  
 c-line.mf  
 c-sigma.mf  
 bold2math.mf  
 barcodes.mf to generate barcodes  
 milstd.tex for logic diagrams  
 milstd.mf

### TeX Programs

This directory contains programs that are of general interest to TeX users in general. Files are located in `pub/tex-programs` for ftp users. Mail users should request files from the `tex-programs` archive.

**dvidoc.patch-sun2**  
                   diffs for sun2 running Sun OS 3.4  
**dvidoc.shar3**  
                   a DVI to character device filter for  
                   Unix BSD systems  
**fig2epic11c.shar**  
                   converts fig code to epic or  
                   eepic files  
**schemetex.sh**  
                   simple support for literate  
                   programming in Lisp. A Unix filter  
                   that translates schemeTeX source  
                   into L<sup>A</sup>TeX source

### TeX

This directory contains style files for plain TeX. Files are located in `pub/tex-style` for ftp users. Mail users should request files from the `tex-style` archive.

**declare.tex** macros to allocate local registers  
**ithyphen.tex** hyphen.tex for Italian  
**mssymb.tex** the definitions for the symbols  
                   in the two "extra symbols" fonts  
                   created at the AMS  
**scorecard.tex**  
                   prints a baseball scorecard for one  
                   team  
**texpictex.tex**  
                   tpic \special changes to P<sub>1</sub>CT<sub>E</sub>X  
                   ◇ Michael DeCorte  
                   2300 Naudain St. "H"  
                   Philadelphia, PA 19146  
                   mrd@sun.soe.Clarkson.edu  
                   Bitnet: mrd@clutz

## Site Reports

### Data General site report

Bart Childs

We have now installed TeX 2.98 and the rest of the changes that have been made to the sources at Stanford since the first of the year. As usual, change the revision number, tangle, compile, ...

The new Data General printers are a considerable improvement over the previous ones. The new ones based upon the Canon engine do not have arbitrarily small limits for downloaded fonts. This driver should work well for the vanilla Canon printer.

We are in the process of rewriting these drivers in CWEB. It will be interesting to perform some timings to see if we can get an improved throughput. These drivers are descendants of dvitype.

---

### Prime 50 Series Site Report

John M. Crawford

We've recently updated our TeX distribution tape to keep up with the latest revisions of software coming from Stanford and friends. This includes updates to TeX, METAFONT, utility programs, and METAFONT sources, as well as a rebuild of some of the METAFONT fonts. The L<sup>A</sup>TeX and A<sub>M</sub>S-TeX source files have been updated. Further updates to this software can be quickly incorporated onto our tape, as our Internet network facilities allow us quick, easy access to the various software repositories.

Updated versions of various device drivers have also been incorporated into our new tape spins, thanks to contributions by some of our friendly off-site TeX and Primos users. We now also have available a version of TeX with greatly expanded memory arrays, by locally incorporating Bart Child's 64 Bit TeX work into our TeX port.

## Typesetting on Personal Computers

### “Free” T<sub>E</sub>X Software for IBM PCs

Jon Radel

Since there have recently been several confusing mentions of the disk copying service I supply to the T<sub>E</sub>X community, I would like to take this opportunity to clarify matters a bit and call attention to my service for those people who missed those mentions. I make copies of a variety of material of use for running T<sub>E</sub>X on an IBM PC or clone. The charge is nominal — to cover my expenses in gathering the material — if you supply the floppies and a return mailer. I can also supply the disks if you prefer to simply send money. I have, at the moment, two ports of T<sub>E</sub>X itself, one of METAFONT, Nelson Beebe’s DVI drivers as well as some other drivers and previewers, the L<sup>A</sup>T<sub>E</sub>X-style collection, back issues of T<sub>E</sub>Xhax and T<sub>E</sub>XMAG, and a variety of other interesting material. I make an effort to carry the most recent version of programs, but I can make no guarantees as I am in part dependent on the authors to let me know about new versions.

I would prefer that you send all mail about this software to me at Jon Radel, P. O. Box 2276, Reston, VA 22090. To get the details on ordering, and the current list of what I have, please send a self-addressed envelope. Attach 45 cents postage in the U.S. Outside the U.S., send International Reply Coupons, 2 for Canada and Mexico, 4 for elsewhere, or, if more convenient for you, US\$2.25.

Incidentally, if you have created any software of use to someone using T<sub>E</sub>X on an IBM PC, I would be most interested in hearing about it if you are willing to give me permission to distribute it.

◊ Jon Radel  
P. O. Box 2276  
Reston, VA 22090  
jonradel@icecream.princeton.edu

### Public Domain T<sub>E</sub>X for the Mac

Andrew Trevorrow

OzT<sub>E</sub>X 1.0 is a public domain version of T<sub>E</sub>X for the Macintosh. It aims to provide a standard T<sub>E</sub>X environment that can be easily extended or customized. People with access to T<sub>E</sub>X on some other computer should feel right at home using OzT<sub>E</sub>X, particularly those who use P<sub>S</sub>PRINT and DVItOVDU on a VAX/VMS or UNIX mainframe.

#### A brief description

Here’s a quick look at OzT<sub>E</sub>X’s major features:

- The complete distribution requires ten 800K disks. Five of these are full of PK files (for a 300dpi write-black laser printer such as the Apple LaserWriter). Another two disks contain the entire source code. OzT<sub>E</sub>X is written in Modula-2 under MPW (Macintosh Programmer’s Workshop).
- The OzT<sub>E</sub>X application includes T<sub>E</sub>X (actually INIT<sub>E</sub>X so users can create their own format files), a DVI page previewer and a PostScript driver that can send output to the current printer or to a text file.
- The three most popular formats are supplied: Plain, L<sup>A</sup>T<sub>E</sub>X and A<sub>M</sub>S-T<sub>E</sub>X.
- OzT<sub>E</sub>X reads standard TFM and PK files and reads and writes standard DVI files.
- The previewer can cope with just about any DVI file you’re ever likely to create, including those generated by another T<sub>E</sub>X system. Have you ever wondered what trip.dvi (the DVI file created by Knuth’s trip test) looks like?
- The application includes a Help menu which you can easily extend or modify.
- A configuration file is read when starting up and controls much of OzT<sub>E</sub>X’s default behaviour. This simple text file can be edited to suit your particular requirements. Some of the parameters you can specify include the printer resolution, the paper dimensions, a list of the formats that appear in the T<sub>E</sub>X menu, and a list of all TFM file names for printer-resident PostScript fonts.
- A 22-page user guide is supplied, including its L<sup>A</sup>T<sub>E</sub>X source. By the time you read this article I should also have finished a system guide aimed at programmers who’d like to modify OzT<sub>E</sub>X.

It’s not all good news however. There is still plenty of room for improvement:

- There is no integrated text editor. OzTeX is distributed with  $\Sigma$ Edit, a public domain DA editor written by Leonard Rosenthal.
- OzTeX requires a PostScript printer.
- $\backslash$ special handling is fairly unsophisticated. OzTeX allows the inclusion of a PostScript file along with optional code prefixed to the file. There is currently no support for previewing PICT or EPSF files.
- Previewing DVI pages is not as fast as I'd like, particularly on a Mac Plus.

Future development of OzTeX is likely but will occur at a fairly sedate pace unless I can find people prepared to help with the programming or provide financial support. Send your bug reports, comments and offers of help to the address shown at the end of this article.

### Where to get OzTeX

The following people have volunteered to help distribute OzTeX. Please get in touch with the person nearest you. By the time you read this article it is likely that OzTeX will also be available electronically from various Mac archive sites. People without access to email should try their local Mac user group.

In Australia and New Zealand:

addie@rhea.trl.oz	Ron Addie, Melbourne
keady@advax.uwa.oz	Grant Keady, Perth
rks105@phys6.anu.oz	Russell Standish, Canberra
ccc032u@aucc4341.aukuni.ac.nz	R. Fulton, Auck.

In the USA:

c3ar@zaphod.uchicago.edu	Walter Carlip, Chicago
tnieland@aamrl.af.mil	Ted Nieland, Dayton
spencer@cis.ohio-state.edu	S. Spencer, Columbus

In the UK and Europe:

abbott@aston.ac.uk	Peter Abbott, UK
texline@vaxa.cc.ic.ac.uk	Malcolm Clark, UK
nikunen@cc.helsinki.fi	Martti Nikunen, Helsinki

I'd like to hear from people interested in distributing OzTeX in other countries. Here's how to get in touch:

Andrew Trevorrow  
 Kathleen Lumley College  
 North Adelaide, SA, 5006, Australia  
 Telephone: (08) 267 1060  
 Email: atrevorrow@g.ua.oz (ACSnet)

## Tutorials

### $\backslash$ string and $\backslash$ csname

Stephan v. Bechtolsheim

This article discusses  $\backslash$ string and  $\backslash$ csname to convert back and forth between strings and tokens. To control loading macro source files in a convenient way, I will show an application of  $\backslash$ csname. I will also discuss cross referencing which relies on  $\backslash$ csname.

### Converting Tokens to Strings, $\backslash$ string

" $\backslash$ string <token>" causes TeX to read the token <token> following  $\backslash$ string without expansion. Subsequently <token> is replaced by a string representing it. Let me start with some examples.

1.  $\{\backslash\text{tt}\backslash\text{string}\backslash\text{hskip}\}$  prints  $\backslash\text{hskip}$ .
2.  $\{\backslash\text{tt}\backslash\text{string}\$\}$  prints \$.
3.  $\{\backslash\text{tt}\backslash\text{string}\$\}$  prints \\$.
4.  $\{\backslash\text{tt}\backslash\text{string}\{\}$  prints {.
5.  $\{\backslash\text{tt}\backslash\text{string}\}\}$  prints }.

Also note:

1. The escape character printed in the previous examples is the backslash. Any other character could be printed by assigning a different character code to  $\backslash$ escapechar. The default is obviously  $\backslash$ escapechar =  $\backslash$ , which assigns the character code of the backslash. If you change  $\backslash$ escapechar to a negative value, then no escape character is printed:  
 $\backslash$ escapechar = -1  $\backslash$ string $\backslash$ xx prints xx.
2. There is an important difference between 'xx' entered as an ordinary string and 'xx' generated using  $\backslash$ string as just shown. All characters generated by  $\backslash$ string have the category code 12 ("other"), whereas 'x' ordinarily has category code 11 ("letter").
3. Observe the use of the typewriter font ( $\backslash$ tt). If you use the roman font and simply write  $\backslash$ string $\backslash$ hskip the output reads "hskip and *not*  $\backslash$ hskip, as expected. The reason for this is that the roman font contains an opening double quote in the position where the typewriter font contains a backslash.
4.  $\backslash$ string converts only the token following it into a string. For instance, to print two consecutive \$\$ you have to repeat  $\backslash$ string and enter  $\{\backslash\text{tt}\backslash\text{string}\$\backslash\text{string}\$\}$ . If you enter only  $\{\backslash\text{tt}\backslash\text{string}\$\$\}$  the first dollar sign is

printed due to `\string`, the second one causes `TeX` to enter math mode.

5. An important application of `\string` is to write control sequences to a file using `\write`. Any control sequence which should be written to a file (instead of being expanded) must be prefixed by `\string`. `\noexpand` can also be used.

### Converting Strings into Tokens,

`\csname ... \endcsname`

**General Discussion.** The `\csname` instruction is, in a certain sense, the inverse operation of `\string`. It converts a sequence of *characters* into one token. Observe that I said “characters” and not “letters.” Using `\csname` allows you to build names for tokens that contain nonletter characters such as digits. The ordinary way to write control sequences restricts the user to control words (the escape character followed by any number of letters, but letters only) and control symbols (the escape character followed by one and only one nonletter character).

The `\csname` control sequence is applied as follows. After `\csname`, list the characters naming the token. You also may use macros, but only those which expand to characters. The sequence of characters forming the name of the token is terminated by `\endcsname`.

Here is an example. To name the token “`\?-a*17.g`” write

```
\csname ?-a*17.g\endcsname
```

In the rest of this article, please allow a certain looseness with respect to notation. Ordinarily, if you see a piece of `TeX` source like `\?-a*17.g`, you would interpret this as `\?` followed by 7 single character tokens. In this article, it stands for one single token.

Depending upon the context of the above example, `TeX` may try to expand the token named and will have to be able to find a corresponding macro definition (or any other type of definition). Here is how one can define such a token.

```
\expandafter\def
  \csname ?-a*17.g\endcsname{%
    Replacement text of macro.
  }
```

The `\expandafter` suppresses the `\def` temporarily to allow `TeX` to compute the name of the token `\?-a*17.g`. Then `\def` is re-inserted in front of this token. The macro definition now proceeds as any other macro definition. If `\expandafter` were

omitted, `TeX` would define a macro with the name `\csname`.

As mentioned before it is legal to call a macro inside a `\csname ... \endcsname` sequence as long as the macro expands to characters only. Counter-registers can also be used:

```
\def\xx{ABC}
\count0 = 4
\csname ZZ-\the\count0-\xx\endcsname
```

This example is equivalent to forming the same token using `\csname ZZ-4-ABC\endcsname`.

I will later discuss two applications of `\csname`. One will define a macro `\InputD` to load macro source files, and the other uses `\csname` for cross-referencing macros.

**`\csname` and `\relax`.** Assuming that *no* preceding definition for `\xx` is given, when `TeX` executes the following code:

```
\csname xx\endcsname
\xx
```

You may be surprised to see that the first line does *not* generate an “undefined control sequence” error, whereas the second line does. The reason is that *undefined* tokens generated by `\csname` are made equivalent to `\relax` by `TeX`. The above code fragment is therefore equivalent to:

```
\relax
\xx
```

On the other hand, in the following examples:

```
\def\xx{This is fun}
\csname xx\endcsname
\xx

\expandafter\def\csname xx\endcsname{%
  This is fun%
}
\csname xx\endcsname
\xx
```

“This is fun” is printed four times.

In other words, if a token named using `\csname` is undefined, it is equivalent to `\relax`. This fact will be used in definition of `\InputD` (next section).

### `\InputD`: Loading Macros Conveniently

**The Problem.** I personally like to divide my macro sources into many small macro source files. I load only those macro source files that I really need. Here is a problem frequently encountered in this context:

1. Assume that at the top of a main source file, `main.tex`, you load macro file `A.tex`. This

macro file in turn uses macros from another macro source file `B.tex`.

2. Assume in addition that you load macro source file `C.tex` in `main.tex`, and that `C.tex` also uses macros from `B.tex`.

If you load both `A.tex` and `C.tex` in your main source file, and both `A.tex` and `C.tex` load `B.tex` using `\input`, then, of course, `B.tex` will be loaded twice, which is undesirable.

**Using `\InputD`.** I will now explain how to define a macro `\InputD` having one argument, the name of a file, which loads the file only if the file was not loaded before. In other words, in `A.tex` and `C.tex` you request `B.tex` to be loaded via `\InputD{B.tex}`. Only the very first time is `\InputD{B.tex}` equivalent to `\input B.tex`; subsequent times, `\InputD{B.tex}` does nothing (actually, in the macro definition below, a message is generated saying that `B.tex` was not loaded again). The macro `\InputD` does all the bookkeeping.

Note that you must always use this macro to load macro source files in order to get the effects described here. After using the ordinary `\input B.tex` (bypassing the bookkeeping of `\InputD`), `TEX` has no record of the fact that `B.tex` was already loaded, and a later occurrence of `\InputD{B.tex}` will cause `TEX` to load `B.tex` again if this is the first call of `\InputD` with argument `B.tex`.

**The Workings of `\InputD`.** When this macro is called, its argument, a file name, will be used to form a token as follows. A prefix, `InputD-`, and the file name will be concatenated. For instance, `\InputD{B.tex}` causes `TEX` to form the token `\InputD-B.tex`. A test is then performed to determine whether this token is already defined. If it is (which will mean the file `B.tex` is already loaded), nothing is done. On the other hand, if this token is *undefined* (this is true only the very first time `\InputD` is called with argument `B.tex`), the macro will *define* token `\InputD-B.tex` and load file `B.tex`. Any subsequent call `\InputD{B.tex}` will find the token `InputD-B.tex` defined. The actual definition of this token is irrelevant. The macro `\InputD` simply defines the token to expand to nothing (i.e., the replacement text is empty).

I think that using this macro offers a very flexible and powerful approach to maintaining lots of macro source files. There is no longer a need to do any bookkeeping of which source files have been loaded: if you need a macro source file, load it using `\InputD`. Source files will be loaded only if they have not been loaded before.

**The Definition of `\InputD`.** The definition of the `\InputD` macro is amazingly simple. Note that `\ifx` absorbs without expansion the two tokens following it, and then compares the two tokens. `\expandafter` is used first to compute the token `InputD-B.tex` (assuming the argument is `B.tex`), and then the `\ifx` compares this token with `\relax`. As noted before, if the token is undefined, it is equivalent to `\relax`.

```

\def\InputD #1{%
  \expandafter\ifx
    \csname InputD-#1\endcsname
    \relax
    % Equivalent to \relax: not
    % defined before! Define now.
  \expandafter\def
    \csname InputD-#1%
      \endcsname{}%
    % Read in macro source file.
  \input #1
  \else
    % Loaded already.
    % Print message.
    \message{\string\InputD:
      file "#1" was loaded
      before.}%
  \fi
}

```

### Cross-Referencing Macros

The following discussion is more complicated than the previous application for `\csname`. This is a brief sketch only (see my book "TEX in Practice" for further information).

**The User Interface.** Let me first explain how you can use the cross-reference macros which I will present later. The following should be familiar to every user of `LATEX`, the main difference being that I use macros which begin with capital letters. To identify document entities such as chapters, sections, figures, etc., *labels* are used. Such labels consist of arbitrary characters like `f-structure` for a figure describing the structure of some piece of equipment.

The user has the following three macros at her/his disposal (all three have one parameter, a label):

1. `\Label` is used to define a label. For instance, `\Label{f-structure}` labels the figure mentioned above, associating a symbolic reference ("`f-structure`") with an appropriate numeric reference (say, "3.5").

2. `\Ref` expands to the figure number of the figure whose label is given as an argument. If you want to print some text like “see Fig. 3.5,” then you would *not* enter “see Fig.~3.5,” because you would have to change this text if the figure number changes (for instance, to 3.6, because you have inserted another figure before this figure). Instead you enter “see Fig.~\Ref{f-structure},” and let T<sub>E</sub>X do the work.
3. `\PageRef` expands to the page number of the figure labeled by the name you provide. Again this page number will automatically change, if the figure migrates to a different page due to modifications of the text. Your input might read “p.~\PageRef{f-structure}” and print “p. 67” if that is the page where the figure with the specified label is placed.

### Retrieving the Cross-Reference Information.

The cross-reference information is read in at the very beginning of a T<sub>E</sub>X job (before any “real” text processing is started) from a *label file*. In the case of L<sup>A</sup>T<sub>E</sub>X, this label information is stored in `.aux` files. We will soon discuss how this information was written to the label file in the first place.

Such label files consist of calls to a macro `\NewLabel`. This macro has three arguments: *label name*, *number* of the entity and *page number* of the entity. So in the above case one call contained in this label file reads as follows:

```
\NewLabel{f-structure}{3.5}{67}
```

`\NewLabel`, when called, will in turn define two macros, one called `\REF-f-structure`, which expands to the number of this figure, and one called `\PAGEREF-f-structure`, which expands to the page number. Here is the definition of `\NewLabel`.

```
\def\NewLabel #1#2#3{%
  \expandafter\def
    \csname REF-#1\endcsname{#2}%
  \expandafter\def
    \csname PAGEREF-#1\endcsname{#3}%
}
```

`\Ref` and `\PageRef` simply retrieve what `\NewLabel` has stored:

```
\def\Ref #1{\csname REF-#1\endcsname}
\def\PageRef #1{%
  \csname PAGEREF-#1\endcsname}
```

Note that a definition of `\Ref` or `\PageRef` would normally be augmented by an `\ifx` test along the lines of the definition of `\InputD` to print a warning message in the case of an undefined label.

**Generating the Label File.** From what we discussed so far you know that the label file is read in at the very beginning of a T<sub>E</sub>X run. Therefore *all* cross-references printed in the text are based on the information of the *previous* run. They do *not* take into account any changes occurring to those labels due to changes in the text during the current run. The same file name is used in the current run both for reading the old label file and for writing new label information in the new version of the label file.

In the definition of `\Label` below it is assumed that `\TheFigureNumber` produces the current figure number. This macro has one parameter which is the name of the label. `\LabelStream` is assumed to be the stream for writing the label file.

```
\def\WriteLab{\write\LabelStream}
\def\Label #1{%
  \edef\LabelTemp{%
    \noexpand\string
    \noexpand\NewLabel
      {#1}{\TheFigureNumber}}
  \expandafter\expandafter\expandafter
    \WriteLab\expandafter{%
      \LabelTemp{\the\pageno}}%
}
```

**Features Not Discussed.** I only tried to sketch here how cross-referencing macros can be implemented. The above macros are far from complete. The following details were ignored.

1. Opening and closing the label file. Reading in the label file in the very beginning.
2. Printing a warning message if two figures are accidentally labeled by the same label.
3. Using labels to label other entities like chapters, sections, tables, etc.
4. Printing a warning message if label definitions as generated during the current run differ from label definitions of the previous run, resulting in possibly wrong cross-references and requiring processing a document for a second time.

### Concluding Remarks

This article was derived from my book “T<sub>E</sub>X in Practice” (the original title was “Another Look at T<sub>E</sub>X”). The book will be published by Springer in October of this year.

◇ Stephan v. Bechtolsheim  
 Integrated Computer Software, Inc.  
 2119 Old Oak Drive  
 West Lafayette, IN 47906  
 svb@cs.purdue.edu



## Macros

### DDA Methods in T<sub>E</sub>X

David Salomon

Several macros are presented here that use DDA methods to generate lines, circles, and ellipses. They are all based on the idea that a curve can be drawn in T<sub>E</sub>X by moving a dot in small steps and repeatedly typesetting it. This idea was originally suggested by Hendrickson [1] for straight lines, and extended by Cameron [2] for wiggly lines. L<sup>A</sup>T<sub>E</sub>X users also have line and circle macros available, but those are limited to certain slopes and diameters.

The macros presented here generate lines, circles, and ellipses, using DDA (Digital Differential Analyzer) methods. DDA is a general name for methods that generate geometric shapes using simple arithmetic operations, and integers. No multiplication, division, square root, or floating-point numbers are used. Typically, a DDA method works by moving along the curve in small steps, calculating the coordinates of the next point  $(x_{i+1}, y_{i+1})$  either as simple functions of the current point  $(x_i, y_i)$ , or using a parametric representation of the curve. Thus either  $x_{i+1} = f(x_i, y_i)$  and  $y_{i+1} = g(x_i, y_i)$ ; or  $(x, y) = f(\phi)$ .

The first macro uses the *Quadrantal DDA* method [3] to produce straight lines of any slope. The second macro is a T<sub>E</sub>X implementation of the *Octantal DDA* method [3], which is somewhat more involved but produces finer lines. The third macro implements *Bresenham's algorithm* [4] for circles; and the last two macros, for ellipses, are based on the parametric equation of these curves. In most of the cases above, the precise shape of the curve depends on the size of the basic step, which is the value of the `\dimen` variable `\step`. It is recommended to first experiment with the macros using `\step=1pt`, just to see how the dot is moved for any given curve. For production purposes, however, it is better to set `\step=.25pt`, which produces a small enough step size such that, in a 300 dpi output, curves look pretty smooth. For higher resolution outputs, the step size should be made even smaller. Unfortunately, making the step size too small, or generating long curves, may result in the dreaded (and, alas, familiar) message:

```
! TeX capacity exceeded,
  sorry [main memory size=65536].
```

The ellipse macros have another potential problem. Large ellipses may cause an arithmetic overflow message, due to T<sub>E</sub>X's limited capacity.

### The Quadrantal DDA Method

Macro `\quadr` typesets a slanted line by using the quadrantal DDA method. It works in any mode and does not move the reference point.

The macro has 2 parameters,  $\Delta x$  and  $\Delta y$ , which are the horizontal and vertical projections of the line, respectively. Since the line starts at the current reference point, the two parameters can also be viewed as the coordinates of the endpoint of the line (relative to the reference point). The parameters can be specified in any valid T<sub>E</sub>X dimension, so expansions such as

```
\quadr -13pt 5in
\quadr 25pc -5mm
\quadr 3cc 3dd
```

are all valid. Note the percent signs '%' at the end of certain macro lines. They are important because T<sub>E</sub>X converts an end of line to a space, but we don't want such spaces to get typeset (try eliminating some of the '%' to see what happens).

To understand the principle of the quadrantal DDA method, consider the case where both  $\Delta x$  and  $\Delta y$  are positive. The line should go up and to the right from the current reference point. The method works by typesetting a dot at the reference point, then moving it, by the basic step, either up or to the right (but not in both directions), typesetting it again, and looping, until the dot has been moved a distance of  $\Delta x$  in the  $x$  direction, and a distance of  $\Delta y$  in the  $y$  direction.

Figure 1 shows two such lines, one almost horizontal and the other, at 45°. Each dot has been magnified to a small box. Note how the principle, of moving either to the right or up, creates the line as a number of *overlapping* segments. This causes the line to appear thicker than it should be. If either  $\Delta x$  or  $\Delta y$  is negative, the dot has to be moved to the left or down. Our algorithm thus has four parts—macro `\doloopA` is used if  $\Delta x \geq 0$  and  $\Delta y \geq 0$  ( $0^\circ \leq \text{slope} \leq 90^\circ$ , the first quadrant); `\doloopB` is used if  $\Delta x < 0$  and  $\Delta y \geq 0$  ( $90^\circ < \text{slope} \leq 180^\circ$ , the second quadrant); and so on.

The decision in what direction to move is based on the value of the `\count` variable `\diff`. `\diff` is initially set to  $-0.5\Delta x$  and is either decremented by  $\Delta x$  (if a decision is made to move the dot up), or incremented by  $\Delta y$  (if the dot is to be moved to the right). By the time the dot gets all the way to the end point of the line, the ratio between the number

of times it has been moved up and the number of times it has been moved to the right is  $\Delta y/\Delta x$ , which is the slope of the line. The algorithm for the first quadrant is therefore:

```
x := 0; y := 0; diff := -\Delta x/2;
repeat
  plot(x, y);
  if diff > 0
    y := y + 1; diff := diff - \Delta x;
  else
    x := x + 1; diff := diff + \Delta y;
until x = \Delta x & y = \Delta y;
```

A simple example is a line with  $\Delta x = 6$  and  $\Delta y = 2$ . The algorithm above iterates 9 times as summarized in the table.

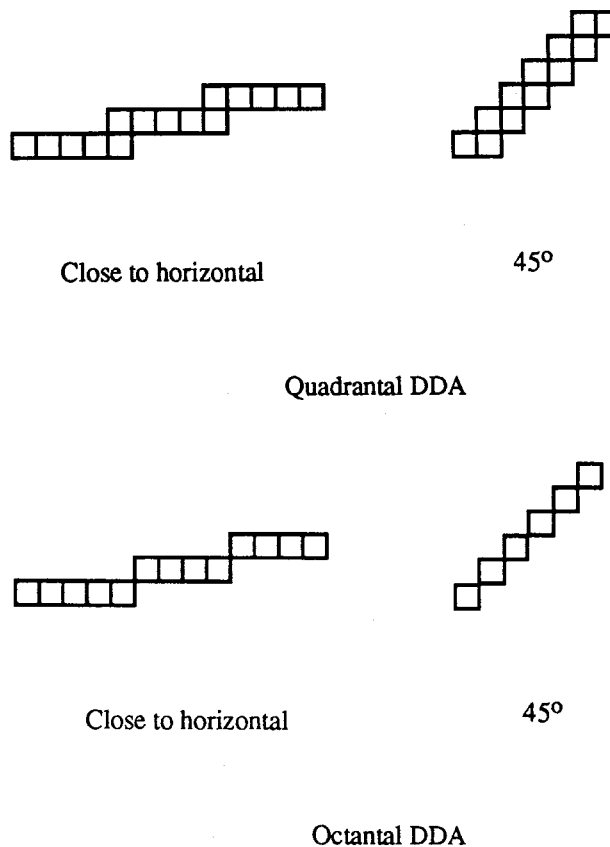
step	x	y	diff	
			before	after
1	0	0	-3	-3 + 2
2	1	0	-1	-1 + 2
3	1	1	1	1 - 6
4	2	1	-5	-5 + 2
5	3	1	-3	-3 + 2
6	4	1	-1	-1 + 2
7	4	2	1	1 - 6
8	5	2	-5	-5 + 2
9	6	2	-3	

For more information on this method, see reference [3].

Macro `\quadr` uses the 4 macros `\doloopA`, ..., `\doloopD`. However, only one of them is expanded, depending on the slope of the desired line. Macro `\point` typesets a dot by placing it in a box of zero dimensions (see page 389 of [5]). Macro `\point` and a number of registers are used by both the quadrantal and octantal methods; a user creating a file of macros for either method must include the following code.

```
1. \newdimen\deltax \newdimen\delthey
2. \newcount\diff
3. \newdimen\stepx \newdimen\stepy
4. \newdimen\step
5. \newif\ifmore
6. %
7. \def\point#1#2{% keep this percent sign!
8.   \vbox toOpt{\kern-#2
9.     \hbox toOpt{\kern#1.\hss}\vss}%
10.  \ifvmode\nointerlineskip\fi}
```

**Exercise 1:** Look carefully at the way macro `\point` generates boxes. What is the depth of those boxes?



**Figure 1. Details of Quadrantal and Octantal Lines**

Note that `\nointerlineskips` are inserted between the dots when `TeX` is in vertical mode. This avoids the interline glue which otherwise is automatically generated. As a result, macro `\quadr` does not move the reference point and, after each expansion, the user should decide whether to move it, and by how much.

```
1. \def\quadr#1 #2 {% keep this % sign!
2.   \deltax=#1 \delthey=#2
3.   \stepx=Opt \stepy=Opt
4.   \ifdim\deltax<Opt
5.     \ifdim\delthey<Opt \doloopC
6.     \else \doloopB \fi
7.   \else
8.     \ifdim\delthey<Opt \doloopD
9.     \else \doloopA \fi
10.  \fi} % end of macro quadr
11. \def\doloopA{%
12.   \ifdim\deltax>\delthey \diff=-\deltax
13.   \else \diff=\delthey \fi
14.   \divide\diff by 2
```

```

15. \loop
16. \ifnum\diff>0
17.   \advance\ystep by \step
18.   \advance\diff by-\deltax
19. \else
20.   \advance\xstep by \step
21.   \advance\diff by \deltay
22. \fi
23. \point{\xstep}{\ystep}%
24. \morefalse
25. \ifdim\xstep<\deltax \moretrue\fi
26. \ifdim\ystep<\deltay \moretrue\fi
27. \ifmore\repeat}
28.     % end of loop for 1st quadrant
29. %
30. \def\doloopB{%
31.   \ifdim-\deltax>\deltay \diff=\deltax
32.   \else \diff=-\deltay \fi
33.   \divide\diff by 2
34. \loop
35. \ifnum\diff>0
36.   \advance\ystep by \step
37.   \advance\diff by \deltax
38. \else
39.   \advance\xstep by-\step
40.   \advance\diff by \deltay
41. \fi
42. \point{\xstep}{\ystep}%
43. \morefalse
44. \ifdim\xstep>\deltax \moretrue\fi
45. \ifdim\ystep<\deltay \moretrue\fi
46. \ifmore\repeat}
47.     % end of loop for 2nd quadrant
48. %
49. \def\doloopC{%
50.   \ifdim-\deltax>-\deltay \diff=\deltax
51.   \else \diff=-\deltay \fi
52.   \divide\diff by 2
53. \loop
54. \ifnum\diff>0
55.   \advance\ystep by-\step
56.   \advance\diff by \deltax
57. \else
58.   \advance\xstep by-\step
59.   \advance\diff by-\deltay
60. \fi
61. \point{\xstep}{\ystep}%
62. \morefalse
63. \ifdim\xstep>\deltax \moretrue\fi
64. \ifdim\ystep>\deltay \moretrue\fi
65. \ifmore\repeat}
66.     % end of loop for 3rd quadrant
67. %
68. \def\doloopD{%
69.   \ifdim\deltax>-\deltay \diff=-\deltax
70.   \else \diff=-\deltay \fi
71.   \divide\diff by 2
72. \loop
73. \ifnum\diff>0
74.   \advance\ystep by-\step
75.   \advance\diff by-\deltax
76. \else
77.   \advance\xstep by \step
78.   \advance\diff by-\deltay
79. \fi
80. \point{\xstep}{\ystep}%
81. \morefalse
82. \ifdim\xstep<\deltax \moretrue\fi
83. \ifdim\ystep>\deltay \moretrue\fi
84. \ifmore\repeat}
85.     % end of loop for 4th quadrant

```

---

### The Octantal DDA Method

Macro `\octnt` typesets a slanted line using *octantal* DDA, a method very similar to quadrantal DDA. The main difference is the way the dot is moved between repeated typesettings. If the line is close to horizontal (its slope is between  $0^\circ$  and  $45^\circ$ ) the dot is moved either to the right, or diagonally (up and to the right). If the line is close to vertical ( $\Delta y > \Delta x$  or the slope is between  $45^\circ$  and  $90^\circ$ ), the dot is moved either up or diagonally. If either  $\Delta x$  or  $\Delta y$  is negative, the directions are changed accordingly.

Fig. 1 shows the way lines appear in this method. The line that is close to horizontal is made of several non-overlapping segments; the  $45^\circ$  line consists of dots laid diagonally. These lines are finer than the quadrantal lines since they consist of fewer dots.

Because of the rules above, the algorithm should distinguish eight orientations of the lines, or eight ranges of the slope (hence the name *octantal*). The main macro, `\octnt`, does exactly that. However, the range  $0^\circ$ – $45^\circ$  (octant 1) is similar to the range  $315^\circ$ – $360^\circ$  (octant 8), so they are both handled by macro `\loopA`. Octants 2, 3 ( $45^\circ$ – $90^\circ$ ,  $90^\circ$ – $135^\circ$ ) are handled by macro `\loopB`, and so on. We thus end up with just four loop macros, instead of eight. Dots are typeset by the same macro, `\point`, used for lines drawn by the quadrantal method.

```

x := 0; y := 0; diff := -\Delta x/2;
repeat
  plot(x, y);
  if diff > 0
    y := y + 1; x := x + 1;

```

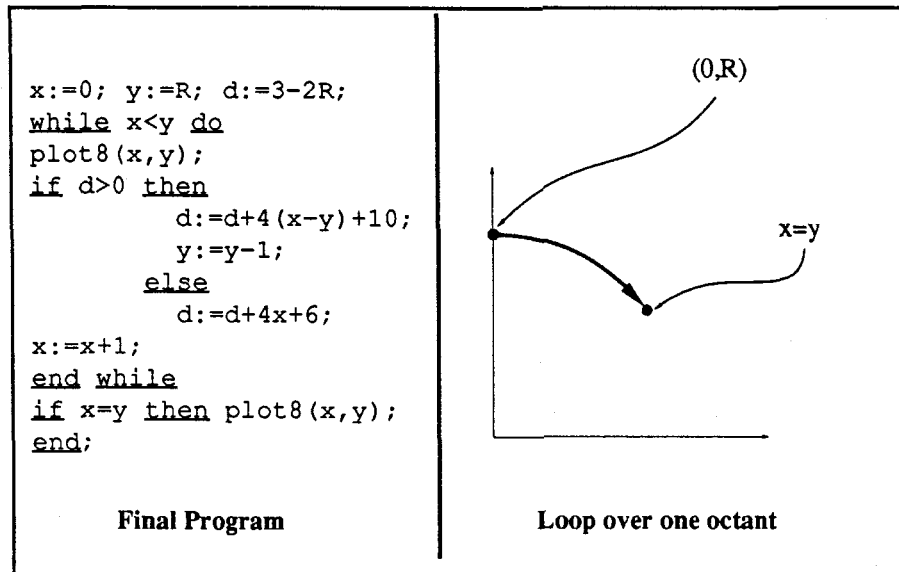


Figure 2. Bresenham's Algorithm for a Circle

```

diff := diff - Δx + Δy;
else
  x := x + 1; diff := diff + Δy;
until x := Δx;

```

And we illustrate the method with the previous example; a line with  $\Delta x = 6$  and  $\Delta y = 2$ . This time the algorithm iterates only 7 times, producing a finer line.

step	x	y	diff	
			before	after
1	0	0	-3	-3 + 2
2	1	0	-1	-1 + 2
3	2	0	1	1 - 6 + 2
4	3	1	-3	-3 + 2
5	4	1	-1	-1 + 2
6	5	1	1	-1 - 6 + 2
7	6	2	-3	

```

1. \newdimen\Absx \newdimen\Absy
2. \newdimen\Xstep \newdimen\Ystep
3. %
4. \def\octnt#1 #2 {% keep this percent sign
5. \deltax=#1 \deltay=#2
6. \xstep=Opt \ystep=Opt
7. \ifdim\deltax<Opt \Absx=-\deltax
8. \else \Absx=\deltax \fi
9. \ifdim\deltay<Opt \Absy=-\deltay
10. \else \Absy=\deltay \fi
11. \ifdim\deltax<Opt
12. \ifdim\deltay<Opt
13. \Xstep=-\step \Ystep=-\step
14. \ifdim\Absx>\Absy \loopD

```

```

15. \else \loopC \fi
16. % octants 5 (loopD) & 6 (loopC)
17. \else
18. \Xstep=-\step \Ystep=\step
19. \ifdim\Absx>\Absy \loopD
20. \else \loopB \fi
21. % octants 4 (loopD) & 3 (loopB)
22. \fi
23. \else
24. \ifdim\deltay<Opt
25. \Xstep=\step \Ystep=-\step
26. \ifdim\Absx>\Absy \loopA
27. \else \loopC \fi
28. % octants 8 (loopA) & 7 (loopC)
29. \else
30. \Xstep=\step \Ystep=\step
31. \ifdim\Absx>\Absy \loopA
32. \else \loopB \fi
33. % octants 1 (loopA) & 2 (loopB)
34. \fi
35. \fi} % end of macro \octnt
36. %
37. \def\stepx{\advance\xstep by \Xstep
38. \advance\diff by \Absy}
39. \def\stepy{\advance\ystep by \Ystep
40. \advance\diff by -\Absx}
41. %
42. \def\loopA{% loop for octants 1 & 8
43. \diff=-\Absx \divide\diff by 2
44. \loop
45. \ifnum\diff>0
46. \stepx \stepy
47. \else \stepx

```

```

48. \fi
49. \point{\xstep}{\ystep}%
50. \morefalse
51. \ifdim\xstep<\deltax \moretrue\fi
52. \ifmore\repeat}
53.      % end of loop for octants 1 & 8
54. %
55. \def\loopB{%      loop for octants 2 & 3
56. \diff=\Absy \divide\diff by 2
57. \ifdim\Absx=\Absy \diff=0 \fi
58. \loop
59. \ifnum\diff>0
60.      \stepy
61. \else \stepx \stepy
62. \fi
63. \point{\xstep}{\ystep}%
64. \morefalse
65. \ifdim\ystep<\deltay \moretrue\fi
66. \ifmore\repeat}
67.      % end of loop for octants 2 & 3
68. %
69. \def\loopC{%      loop for octants 6 & 7
70. \diff=\Absy \divide\diff by 2
71. \ifdim\Absx=\Absy \diff=0 \fi
72. \loop
73. \ifnum\diff>0
74.      \stepy
75. \else \stepx \stepy
76. \fi
77. \point{\xstep}{\ystep}%
78. \morefalse
79. \ifdim\ystep>\deltay \moretrue\fi
80. \ifmore\repeat}
81.      % end of loop for octants 6 & 7
82. %
83. \def\loopD{%      loop for octants 4 & 5
84. \diff=-\Absx \divide\diff by 2
85. \loop
86. \ifnum\diff>0
87.      \stepx \stepy
88. \else \stepx
89. \fi
90. \point{\xstep}{\ystep}%
91. \morefalse
92. \ifdim\xstep>\deltax \moretrue\fi
93. \ifmore\repeat}
94.      % end of loop for octants 4 & 5

```

Note that `\loopA` (octants 1 and 8) repeats while the  $x$ -coordinate of the dot is  $< \Delta x$ . `\loopD` (octants 4 and 5, where  $x$  and  $\Delta x$  are negative), however, repeats while  $x > \Delta x$ . This works since in those octants the line is closer to horizontal. `\loopB`

and `\loopC`, where lines are close to vertical, are similar but compare  $y$  and  $\Delta y$ .

### The Bresenham-Michener DDA Method for Circles

Because of the high symmetry of a circle, it is a particularly easy figure to draw (See [6] for a number of circle drawing methods). The method used here is efficient since it uses only integers and requires only addition, subtraction, and a multiplication by 4. When this method is implemented as a computer program, the multiplication by 4 is usually replaced by a shift. `TeX`, however, cannot shift numbers. The method is described here in two stages. First the basic idea (see algorithm in Fig. 2) is outlined; next, the `TeX` implementation is explained.

The basic idea is to draw a circle of radius  $R$ , centered around the origin, by starting at the top of the circle (point  $(0, R)$ ) and moving, in small steps, along one octant of the circle. Because of the symmetry of a circle, each time a point  $(x, y)$  is calculated on one octant, seven more points—on the seven other octants—can be calculated, which correspond to the original point. They are:  $(-x, y)$ ,  $(x, -y)$ ,  $(-x, -y)$ ,  $(y, x)$ ,  $(-y, x)$ ,  $(y, -x)$ , and  $(-y, -x)$ . The algorithm is a simple loop that starts at point  $(x, y) = (0, R)$  and continues while  $x < y$  (i.e., over one octant). Each time through the loop, the current point (plus the seven corresponding points) is typeset, and the algorithm moves to the next point by incrementing the  $x$  coordinate by `\step` and, from time to time, also decrementing the  $y$  coordinate (by the same `\step`). Variable `\step` has to be assigned a value before `\circle` is expanded.

The only decision that has to be made in each iteration is whether or not to decrement the  $y$  coordinate. This decision involves the auxiliary `\dimen` variable `\d` whose sign determines the action taken. If `\d` is non-negative then  $y$  is decremented. Each time through the loop `\d` is updated. The details of updating `\d` can be found in references [4, 6] or can be obtained by writing to this author.

The `TeX` implementation presented here builds the circle centered on the reference point. The reference point itself is not moved and, after each expansion of the macro, the user may want to move it explicitly, using appropriate skip commands. Macro `\circle` is a simple `\loop` construct that expands a plot macro to plot the current point (actually, eight points), and then calculates the coordinates of the next point. Like the macro `\point` used to plot slanted lines, macro `\plot` typesets a dot enclosed in boxes of zero dimensions

(this is why the reference point is not affected), and in vertical mode, inserts `\nointerlineskip` between boxes to eliminate unwanted interline glue. Again note the percent signs '%' at the end of certain source lines. They are important and have been mentioned earlier.

---

```

1. \newdimen\xc \newdimen\yc
2. \newdimen\d \newdimen\yc
3. \newdimen\unitc
4. %
5. \def\circle#1{\unitc=1pt
6. \xc=0pt \yc=#1 \d=3\unitc
7. \advance\d by-2\yc
8. \loop
9. \ploteight
10. \ifdim\d>0pt
11. \tc=\xc \advance\yc by-\yc
12. \multiply\yc by 4
13. \advance\yc by 10\unitc
14. \advance\d by \tc
15. \advance\yc by-\step
16. \else
17. \tc=4\xc
18. \advance\yc by 6\unitc
19. \advance\d by \tc
20. \fi
21. \advance\yc by \step
22. \ifdim\yc<\yc \repeat}
23. %
24. \def\ploteight{%
25. \plot{\xc}{\yc}\plot{-\xc}{\yc}%
26. \plot{\xc}{-\yc}\plot{-\xc}{-\yc}%
27. \plot{\yc}{\xc}\plot{-\yc}{\xc}%
28. \plot{\yc}{-\xc}\plot{-\yc}{-\xc}}
29. %
30. \def\plot#1#2{%
31. \vbox to0pt{\kern#2
32. \hbox to0pt{\kern#1.\hss}\vss}%
33. \ifvmode\nointerlineskip\fi}

```

---

### Drawing Ellipses

The ellipse macros below accept, as parameters, the semimajor and the semiminor ellipse axes, measured in pt. The first macro generates a canonical ellipse (centered on the reference point with a horizontal major axis); the second one generates an ellipse tilted clockwise  $\theta$  degrees.

Bresenham's method can be generalized to an ellipse (Try to do it! This is exercise 2.), but this does not give good results because of the symmetry of the ellipse, which is not as high as that of a circle. In the case of a circle, it is enough to calculate one octant and copy it over to the other seven. In

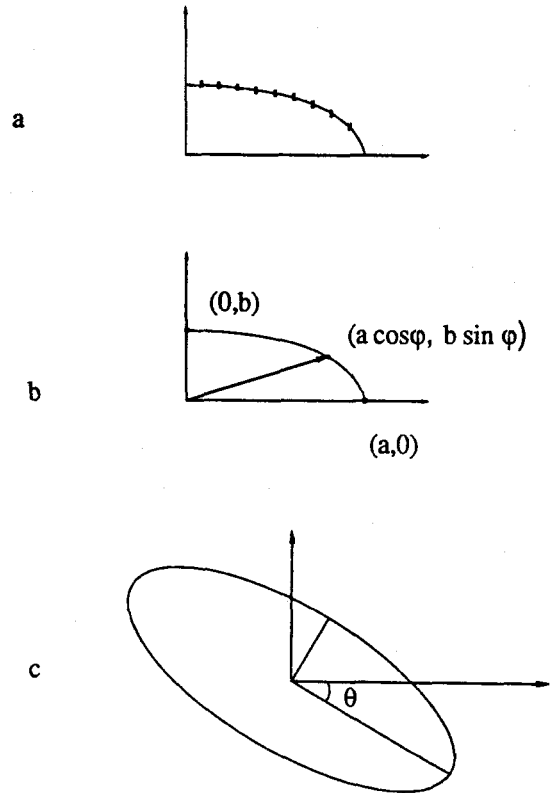


Figure 3. Ellipses

the case of an ellipse, one quadrant, at least, has to be calculated. Bresenham's method is based on looping in equal steps of  $x$ , and this produces good results in the first octant, since that octant has a small slope and does not deviate much from a horizontal line. Looping in equal steps of  $x$  over a quadrant, however, produces dots that are too widely spaced at the end of the quadrant (Fig. 3a), where the ellipse has a large slope.

The method used here to draw an ellipse is well known [7, 8] and is based on the parametric representation of the ellipse:

$$x = a \cos \phi; \quad y = b \sin \phi. \quad \phi = 0 \dots 360^\circ \quad (1)$$

where  $a$  is the semimajor axis and  $b$ , the semiminor one (Fig. 3b). The parameter  $\phi$  is varied (in small steps of  $d\phi$ ), over a quarter of the ellipse, from point  $(a,0)$ , ( $\phi = 0$ ), to point  $(0,b)$ , ( $\phi = 90^\circ$ ).

An important property of the algorithm is that varying  $\phi$  in fixed steps of  $d\phi$  moves the dot along the ellipse in steps that cover variable perimeter sizes. Initially, around point  $(a,0)$ , the step size is small, which is appropriate for that region, where the ellipse has a large slope. As we move along the quadrant toward the final point  $(0,b)$ , the

step size covers larger perimeter increments, again appropriate for this region, where the slope gets smaller.

To demonstrate this property, we derive the differential of Eq. 1.

$$dx = -a \sin \phi d\phi; \quad dy = b \cos \phi d\phi.$$

For the initial steps, where  $\phi$  is close to zero,  $dy \approx b d\phi$  and  $dx$  is close to zero. Toward the end, where  $\phi$  is close to  $90^\circ$ ,  $|dx| \approx a d\phi$  and  $dy \approx 0$ . The perimeter increment is thus initially close to  $b d\phi$  and gets larger as it approaches  $a d\phi$ . Also, the ratio between the initial and final perimeter increments is approximately  $b/a$ , which is the ratio of the two axes of the ellipse. If  $a = b$ , the perimeter increment is fixed, which is appropriate for a circle.

Our method uses the elementary trigonometric identities:

$$\begin{aligned} \sin(x+y) &= \sin x \cos y + \cos x \sin y; \\ \cos(x+y) &= \cos x \cos y - \sin x \sin y. \end{aligned} \quad (2)$$

Using Eq. 1, we start with  $\phi = 0$  and get:

$$\begin{aligned} x_0 &= a \cos 0 = a; & y_0 &= b \sin 0 = 0. \\ x_1 &= a \cos(d\phi); & y_1 &= b \sin(d\phi). \\ x_2 &= a \cos(2d\phi); & y_2 &= b \sin(2d\phi). \end{aligned}$$

And, in general

$$x_i = a \cos(i d\phi) = a \times A_i; \quad y_i = b \sin(i d\phi) = b \times B_i.$$

The DDA nature of the algorithm stems from the fact that we can eliminate the need for calculating  $\sin(i d\phi)$ ,  $\cos(i d\phi)$  for every value of  $i$ . Using Eq. 2, it is possible to express both  $A_i$ ,  $B_i$  as functions of  $A_{i-1}$ ,  $B_{i-1}$  with the result that only  $\sin(d\phi)$ ,  $\cos(d\phi)$  need be known.

$$A_i = \cos(i d\phi) = \cos((i-2)d\phi + 2d\phi)$$

using Eq. 2 yields

$$\begin{aligned} A_i &= \cos((i-2)d\phi) \cos(2d\phi) \\ &\quad - \sin((i-2)d\phi) \sin(2d\phi); \end{aligned}$$

using Eq. 2 again

$$\begin{aligned} A_i &= \cos((i-2)d\phi) [\cos^2(d\phi) - \sin^2(d\phi)] \\ &\quad - 2 \sin((i-2)d\phi) \sin(d\phi) \cos(d\phi); \end{aligned}$$

adding and subtracting the same term

$$\begin{aligned} A_i &= \cos((i-2)d\phi) \cos^2(d\phi) \\ &\quad - \sin((i-2)d\phi) \sin(d\phi) \cos(d\phi) \\ &\quad - \sin((i-2)d\phi) \cos(d\phi) \sin(d\phi) \\ &\quad - \cos((i-2)d\phi) \sin^2(d\phi) \\ &= A_{i-1} \cos(d\phi) - B_{i-1} \sin(d\phi). \end{aligned} \quad (3)$$

And, similarly,

$$B_i = B_{i-1} \cos(d\phi) + A_{i-1} \sin(d\phi). \quad (4)$$

The initial values are  $A_0 = \cos 0 = 1$ ,  $B_0 = \sin 0 = 0$ . Our algorithm can now be expressed as:

```
A := 1; B := 0; C := cos(dφ); S := sin(dφ);
x := a; y := 0;
loop
  plot (x, y) plus three symmetric points
  T := A × C - B × S;
  B := B × C + A × S;
  A := T;
  x := a × A; y := b × B;
while x > 0;
```

This algorithm involves multiplications, and is therefore considerably slower than Bresenham's, but then a circle is just a special case of an ellipse. Needless to say, the ellipse macro below can be used to generate circles. The macro is a  $\text{\TeX}$  implementation of the rules above, with two exceptions:

1. In principle, the user should supply a value for  $d\phi$  and the macro should calculate  $\sin(d\phi)$  and  $\cos(d\phi)$ . However, since those calculations involve fractions, they have to be done, in  $\text{\TeX}$ , with scaled numbers, which is time consuming. As a result, three pairs of  $\sin(d\phi)$  and  $\cos(d\phi)$  are built into the macro, corresponding to  $d\phi$  values of  $2\pi/120$ ,  $2\pi/240$  and  $2\pi/480$ . Those values were selected experimentally, to produce smooth ellipses on a 300 dpi output. On higher resolution output devices, smaller values should be tried, which may result in finer curves. The first pair generates an ellipse by typesetting 120 dots (actually, generating 30 dots and duplicating each 4 times), the second typesets 240 dots and the third, 480. The macro selects one of those pairs, depending on the size of the ellipse.
2.  $\text{\TeX}$  can easily operate on integers but our problem involves real numbers. Such problems are handled in  $\text{\TeX}$  in one of two ways. The first is to use `dimen` variables, which can have non-integer values; the second makes use of scaled integers. Our macro uses the second choice and scales all numbers by a `\scalefactor` of 10000.

The following registers and macro are common to both ordinary and tilted ellipses. Once again, note the similarity of `\plotu` to the earlier `\point` and `\plot` macros.

---

```
1. \newcount\ a \newcount\ A
2. \newcount\ b \newcount\ B \newcount\ T
3. \newcount\ c \newcount\ C
```

```

4. \newcount\s \newcount\S \newcount\t
5. \newcount\x \newcount\y
6. \newcount\scalefactor \scalefactor=10000
7. \newdimen\unit
8. \unit=1pt \divide\unit by \scalefactor
9. %
10. \def\plotu#1#2{%
11.   \vbox to0pt{\kern#2\unit
12.     \hbox to0pt{\kern#1\unit.\hss}\vss}%
13.   \ifvmode\nointerlineskip\fi}

```

The macro for the ellipse.

```

1. \def\ellipse#1 #2 {%
2.   \A=10000 \B=0
3.   \ifnum#1>#2 \a=#1 \b=#2
4.   \else \a=#2 \b=#1 \fi
5. % d\phi is determined according to the
6. % value of the semimajor axis 'a'.
7. \ifnum\a<15
8.   \S=523 \C=9986 % sin & cos of 360/120,
9.   % correspond to 30 increments of d\phi
10. \else
11.   \ifnum\a<40 %over one quarter
12.     \S=262 \C=9997 %For large ellipses,
13.     % here are 60 increments
14.   \else
15.     \S=131 \C=9999 % and, for the largest
16.     % ones, 120 increments
17. \fi \fi
18. %
19. \x=\a \multiply\x by \scalefactor \y=0
20. \loop
21. \plotfour
22. \T=\B \multiply\T by\S
23. \t=\A \multiply\t by\C \advance\t by-\T
24. \T=\A \multiply\T by\S
25. \multiply\B by\C \advance\B by\T
26. \divide\B by \scalefactor
27. \A=\t \divide\A by \scalefactor
28. \x=\a \multiply\x by\A
29. \y=\b \multiply\y by\B
30. \ifnum\x>0 \repeat}
31. %
32. \def\plotfour{%
33.   \plotu{\x}{\y}\plotu{-\x}{\y}%
34.   \plotu{\x}{-\y}\plotu{-\x}{-\y}}

```

Next, we turn to a tilted ellipse, obtained by rotating the canonical ellipse  $\theta$  degrees clockwise. Mathematically, rotating a 2D point  $(x, y)$  is achieved by multiplying it by the rotation matrix

$$\begin{pmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{pmatrix}.$$

Thus the general ellipse point  $(a \cos \phi, b \sin \phi)$  is transformed into

$(a \cos \phi \cos \theta - b \sin \phi \sin \theta, a \cos \phi \sin \theta + b \sin \phi \cos \theta)$ , and the expressions  $x := a \times A$ ,  $y := b \times B$ , used earlier for the coordinates of the next point now become

$$x := a A \cos \theta - b B \sin \theta \quad y := a A \sin \theta + b B \cos \theta.$$

The algorithm for the tilted ellipse differs from the one for the canonical one in three more points:

1. Since a tilted ellipse is not symmetric with respect to the coordinate axes, macro `\Plotfour` cannot duplicate a point as easily as before. A look at Fig. 3c shows that, for each point  $(x, y)$  on one quadrant of the ellipse, point  $(-x, -y)$  is on the diagonally opposite quadrant but points  $(-x, y)$ ,  $(x, -y)$  are not on the ellipse. The macro should therefore calculate one of these points explicitly, using the expressions

$$u := a A \cos \theta + b B \sin \theta;$$

$$w := a A \sin \theta - b B \cos \theta.$$

and plot points  $(u, w)$ ,  $(-u, -w)$ .

2. The loop for the canonical ellipse is terminated when  $x$  reaches zero. This again won't work for the tilted ellipse, so the new macro `\tellipse` uses a new count variable to loop 30, 60, or 120 times, depending on the size of the ellipse. In each iteration, the count variable is decremented and compared to zero.
3. Because of the additional multiplications necessary, numbers cannot be scaled as high as in the previous macro. Trying to scale all numbers with a factor of 10000 causes arithmetic overflow, so reduced scaling is used, resulting in a less precise shape of the ellipse.

The algorithm thus is:

$$A := 1; B := 0; C := \cos(d\phi); S := \sin(d\phi);$$

$$x := a \cos \theta; u := x;$$

$$y := a \sin \theta; w := y;$$

$$\text{count} := 30, 60, \text{ or } 120$$

**loop**

$$\text{plot } (x, y), (-x, -y), (u, w), (-u, -w)$$

$$T := A \times C - B \times S;$$

$$B := B \times C + A \times S;$$

$$A := T;$$

$$LA := a \cdot A; LB := b \cdot B;$$

(with a reduced scale factor)

$$x := LA \cos \theta - LB \sin \theta;$$

$$y := LA \sin \theta + LB \cos \theta;$$

$$u := LA \cos \theta + LB \sin \theta;$$

$$w := LA \sin \theta - LB \cos \theta;$$

$$\text{count} := \text{count} - 1$$



```

while count > 0;
and the macro is:


---


1. \newcount\dphi
2. \newcount\u \newcount\w
3. \newcount\ST \newcount\CT
4. \newcount\LA \newcount\LB
5. %
6. \def\tellipse#1 #2 #3 #4 {%
7.   \A=10000 \B=0
8.   \d=#3pc \ST=\d
9.   \divide\ST by 786 %since 1pc=786432sp
10.  \d=#4pc \CT=\d \divide\CT by 786
11.  \ifnum#1>#2 \a=#1 \b=#2
12.  \else \a=#2 \b=#1 \fi
13. % d\phi is determined according to the
14. % value of the semimajor axis a.
15.  \ifnum\a<15
16.    \S=523 \C=9986 \dphi=31
17.    % sin, cos of 360/120, for 30
18.    % increments of d\phi
19.  \else \ifnum\a<40 % over one quarter.
20.    \S=262 \C=9997 \dphi=61
21.    % For large ellipses, here are 60
22.    % increments
23.  \else
24.    \S=131 \C=9999 \dphi=121
25.    % For the largest ones, 120
26.    % increments
27.  \fi \fi
28. %
29. \x=\a \multiply\x by\CT
30. \multiply\x by 10 \u=\x
31. \y=\a \multiply\y by\ST
32. \multiply\y by 10 \w=\y
33. \loop
34.  \Plotfour
35.  \T=\B \multiply\T by\S
36.  \t=\A \multiply\t by\C
37.  \advance\t by-\T
38.  \T=\A \multiply\T by\S
39.  \multiply\B by\C
40.  \advance\B by\T
41.  \divide\B by \scalefactor
42.  \A=\t \divide\A by \scalefactor
43. %
44. \LA=\A \multiply\LA by\A
45. \divide\LA by 100
46. \LB=\B \multiply\LB by\B
47. \divide\LB by 100
48. %
49. \x=\LA \multiply\x by\CT \u=\x
50. \T=\LB \multiply\T by\ST
51. \advance\x by-\T \divide\x by 10

```

```

52. \advance\u by \T \divide\u by 10
53. \y=\LA \multiply\y by\ST \w=\y
54. \T=\LB \multiply\T by\CT
55. \advance\y by \T \divide\y by 10
56. \advance\w by-\T \divide\w by 10
57. \advance\dphi by-1
58. \ifnum\dphi>0 \repeat}
59. %
60. \def\Plotfour{%
61.  \plotu{\x}{\y}\plotu{\u}{\w}%
62.  \plotu{-\x}{-\y}\plotu{-\u}{-\w}}

```

This method is considerably slower than the ones for lines and circles because of the multiplications involved. It turns out that, even though not all the multiplications can be eliminated, the method can be made a little more efficient. Reference [9] shows how to modify it to include only four multiplications (and four additions) per iteration. The algorithm is:

```

CT := cos θ; ST := sin θ;
CDP := cos dφ; SDP := sin dφ;
A := CDP + SDP × ST × CT × (a/b - b/a);
B := -SDP ((b × ST)2 + (a × CT)2) / (a × b);
C := SDP ((b × CT)2 + (a × ST)2) / (a × b);
D := CDP + SDP × ST × CT × (b/a - a/b);
D := D - (C × B) / A;
C := C / A;
x := a × CT; y := a × ST;
count := 30, 60, or 120;
loop
  plot (x, y), (-x, -y), (u, w), (-u, -w)
  x := x × A + y × B;
  y := x × C + y × D;
  count := count - 1
while count > 0;

```

The reader is encouraged to implement this in T<sub>E</sub>X.

## Appendix

The methods described here give reasonably good output on a typical 300dpi printer. Sometimes, however, high quality output is a must. Here is an idea which produces better looking results. It is, unfortunately, slow and is more liable to exceed T<sub>E</sub>X's capacity.

All the examples above generate the lines and curves by typesetting a period. The period has a small size but is not small enough for high quality results. Using a period from a smaller size font does not help much. It turns out that the width of a period in font cmr10 is 2.77779pt whereas in font cmr5 it is 2.01392pt, almost the same size.

To get dots of smaller sizes, we therefore suggest typesetting a rule (specifically, a `\vrule`) instead of a dot. The height and width of a rule can easily be controlled and our experiments show that a rule of dimensions 0.1pt, combined with a step size of the same dimension, produces fine, smooth lines on a 300dpi laser printer. The only change necessary is to replace the dot with a `\vrule` in macro `\point` as shown below.

```
\def\vr{\vrule height.1pt width.1pt}
\def\point#1#2{% keep this percent sign!
  \vbox to0pt{\kern-#2
    \hbox to0pt{\kern#1\vr\hss}\vss}%
  \ifvmode\nointerlineskip\fi}
\fi
}
```

### Answers to Exercises

1. Zero, since the depth of a dot is zero. This is easy to verify by `\setbox0=\hbox{.}`, `\showthe\dp0`
2. Generalizing Bresenham's method for ellipses is straightforward and produces:

```
x := 0; y := a;
D := (a/b)2; d := 2D + 1 - 2a;
while x < y do
  plot8(x, y);
  if d < 0
    d := d + D(4x + 6);
  else
    d := d + 4D(x - y) + 6D + 4;
    y := y - 1;
  x := x + 1;
end while;
if x = y then plot8(x, y);
end;
```

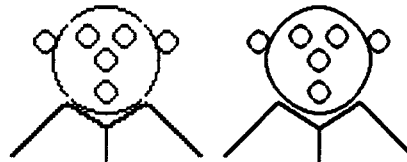
### References

1. Hendrickson, A., *Some Diagonal Line Hacks*, TUGboat 6(2)83-86, (July 1985).
2. Cameron, A. G. W., *Wiggly lines*, TUGboat 6(3)155-156, (Nov. 1985).
3. Artwick, B., *Computer Graphics*, Prentice-Hall, Englewood Cliffs, NJ.: 1985.
4. Bresenham, J. E., *A Linear Algorithm for Incremental Display of Circular Arcs*, Comm. ACM 20(2)100-106(Feb. 1977).
5. Knuth, D. E., *The T<sub>E</sub>Xbook*, Addison-Wesley, Reading, MA.: 1983.
6. Blinn, J. F., *How Many Ways Can You Draw a Circle?*, IEEE Comp. Graphics & Applic. 7(8)39-44(Aug. 1987).
7. Hearn, D., & J. P. Baker, *Computer Graphics*, Prentice-Hall, Englewood Cliffs, NJ.: 1986.

8. Rogers, D. F., & J. A. Adams, *Mathematical Elements for Computer Graphics*, 2nd ed., McGraw-Hill, New York, NY.: 1989.
9. Smith, L. B., *Drawing Ellipses with a Fixed Number of Points*, The Computer J., 14(1)81-86, Feb. 1971.

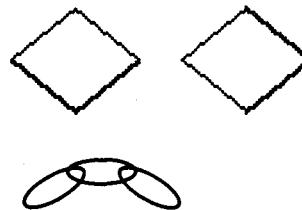
### Tests

Question: What is the difference between these two identical twins?



Answer: The one on the left was done with `\step=1pt`; the one on the right, with `\step=.5pt`

Compare the two diamonds. It is easy to tell which parts are done with quadrantal and which, with octantal DDA.



◇ David Salomon  
California State University,  
Northridge  
Computer Science Department  
School of Engineering and  
Computer Science  
18111 Nordhoff Street  
Northridge, CA 91330  
bccscdxs@csunb.csunb.edu

Editor's note: The methods described in this article might be applicable to a graphics system of the kind sought by David Rogers in his challenge of TUGboat 10#1 (p. 39).

Editor's note: In these macros, the names used by several plain control sequences (`\b`, `\B`, `\c`, `\d`, `\S`, `\t`, `\u`) have been reassigned with `\newdimen`. Beware that these names will remain associated with dimension registers even if `\begingroup ... \endgroup` is used in an attempt to localize their effects.

## Printing Vietnamese characters by adding diacritical marks via T<sub>E</sub>X

Brother Eric Vogel FSC

### Introduction

The technique described here uses T<sub>E</sub>X to produce the diacritical marks for the various vowels needed for Vietnamese. The marks used are: accent grave, accent acute, tilde, question mark and dot (below). New vowels are introduced using the above by placing a hat (above a or e) or breve accent (above a) or by attaching a "beard" (using the breve accent) to o and u. The macros for these definitions are listed below in Table 3.

These macros were developed using PC T<sub>E</sub>X; they will work equally well with any other T<sub>E</sub>X. Other facilities described here, however, are specific to PC T<sub>E</sub>X and to PC-Write.

### Method of producing text with diacritical marks attached

The author of the Vietnamese text enters the text without the marks. The author then edits the text by placing the cursor on the letter needing a diacritical mark and striking a key sequence that produces the mark(s). What appears on the screen is the command for T<sub>E</sub>X to reproduce the correct marks upon printing. For example to produce the Vietnamese word *Măg* the author starts with *Mag* on the screen and the cursor under the a. The author then presses control-k. On the screen appears `\gu`, followed by the original a and g. When the document is T<sub>E</sub>Xed, *Măg* is printed.

### Definitions used by T<sub>E</sub>X

The accents are produced by placing the correct command (i.e. definition) in front of the letter to which one wishes to attach the accent. Because of the additional space needed for o-beard and u-beard, these commands (definitions) contain the needed o or u (O or U); hence if one has text to which accents are being added, it is necessary to delete the o or u (O or U) which is in the text. The author has used PC-Write (Shareware) to define the introduction of these commands using either the control key, the shift key or control-c with an additional letter. Table 2 displays the letters as accented by this program as well as what one types using the editor of PC-Write and what is displayed on the screen. To define these control sequences it is necessary to add to the `ed.def` file of PC-Write the command `!viet.ext` in order that the editor can access the file `viet.ext` which is given in Table 1.

Special attention needed to be paid to producing the diacritical mark of a question mark over letters as the Vietnamese mark does not have a dot under the rest of the question mark. This was accomplished by using METAFONT to generate a new ? in the font `vfont`. Two files are needed and must be copied into appropriate subdirectories of PC T<sub>E</sub>X: `vfont.tfm`, which is copied into the subdirectory `\textfms`, and `vfont.pk`, which is copied into the subdirectory `\pixel\dpi300`. The following code was used in naming the definitions: a indicates accent acute; g indicates accent grave; t indicates tilde; q indicates question mark; d indicates dot (under the vowel); h indicates hat; combinations of the above indicate more than one accent: e.g. `ah` indicates accent acute over hat. The accents that include a "beard" attached to either o or u all have a b in their code. `ob` indicates an o with a beard; `ub` indicates a u with a beard; b can be combined with the other accents given above: e.g. `ba` indicates a beard and an accent acute. The capitals for U and O have special commands (necessitated by the lack of keys on the keyboard). These take advantage of the special two stroke capabilities of PC-Write and are addressed by using control-c followed by a letter. The coding of the commands consists in using `co` to indicate a capital O and `cu` to indicate a capital U (with, of course b for beard) and the possibility of another letter for other accents. For example, `"\cuba"` indicates a capital U, with a beard, with accent acute. Because the author ran out of keys to be redefined using the above method before he ran out of needed keys, two additional keys were redefined: @ and # as shown below.

### Obtaining the files

Upon your request, the author will send you a disk with four files: `vdefq` and `viet.ext` and the two fonts for printing: `vfont.tfm` and `vfont.pk`. The first contains all the macros used to produce the definitions of the diacritical marks via T<sub>E</sub>X. The second is the file needed by PC-Write (to be placed in the `ed.def` file of that software) to define the control sequences which will produce the correct results. Of course the macros can be used with any software that is capable of defining enough keys. The two fonts for printing must be copied into the correct subdirectories as was described above. (Any donations gladly accepted.)

Brother Eric Vogel FSC  
P. O. Box 5150  
Saint Mary's College  
Moraga, Calif. 94575

**Table 1. Key redefining used by PC-Write. Parentheses not included. Found in file: viet.ext.**

064:092,113,117,032 (Redefines @ key to be \qu, question over breve)  
 035:092,117,100,032 (Redefines # key to be \ud, breve over, dot under)  
 c:555, "CAPBEAR"; (Beginning of redefinition of keys via two strokes)  
 556,113, "\cob "; (Ctrl-c, q. Capital O with a beard)  
 556,119, "\cobg "; (Ctrl-c, w. Capital O with a beard and accent grave)  
 556,101, "\coba "; (Ctrl-c, e. Capital O with a beard and accent acute)  
 556,114, "\cobt "; (Ctrl-c, r. Capital O with a beard and tilde)  
 556,116, "\cobq "; (Ctrl-c, t. Capital O with a beard and question mark)  
 556,121, "\cobd "; (Ctrl-c, y. Capital O with a beard and dot below)  
 556,097, "\cub "; (Ctrl-c, a. Capital U with a beard)  
 556,115, "\cubg "; (Ctrl-c, s. Capital U with a beard and accent grave)  
 556,100, "\cuba "; (Ctrl-c, d. Capital U with a beard and accent acute)  
 556,102, "\cubt "; (Ctrl-c, f. Capital U with a beard and tilde)  
 556,103, "\cubq "; (Ctrl-c, g. Capital U with a beard and question mark)  
 556,104, "\cubd " (Ctrl-c, h. Capital U with a beard and dot below)  
 z: "\g " (Ctrl-z. Accent grave)  
 x: "\a " (Ctrl-x. Accent acute)  
 v: "\t " (Ctrl-v. Tilde)  
 b: "\q " (Ctrl-b. Question mark)  
 n: "\d " (Ctrl-n. Dot under letter)  
 m: "\tu " (Ctrl-m. Tilde over breve)  
 a: "\ub " (Ctrl-m. Lower case u with a beard)  
 s: "\bg u\s " (Ctrl-s. u with a beard and accent grave)  
 d: "\ba u\s " (Ctrl-d. u with a beard and accent acute)  
 f: "\bt u\s " (Ctrl-f. u with a beard and tilde)  
 g: "\bq u\s " (Ctrl-g. u with a beard and question mark)  
 h: "\bd u\s " (Ctrl-h. u with a beard and dot below)  
 j: "\u " (Ctrl-j. Accent breve)  
 k: "\gu " (Ctrl-k. Accent grave over breve)  
 l: "\au " (Ctrl-l. Accent acute over breve)  
 q: "\ob " (Ctrl-q. Lower case o with a beard)  
 w: "\bg o\s " (Ctrl-w. o with beard and accent grave)  
 e: "\ba o\s " (Ctrl-e. o with beard and accent acute)  
 r: "\bt o\s " (Ctrl-r. o with beard and tilde)  
 t: "\bq o\s " (Ctrl-t. o with beard and question mark)  
 y: "\bd o\s " (Ctrl-y. o with a beard and dot below)  
 u: "\h " (Ctrl-u. Hat over letter)  
 i: "\gh " (Ctrl-i. Accent grave over hat)  
 o: "\ah " (Ctrl-o. Accent acute over hat)  
 p: "\th " (Ctrl-p. Tilde over hat)  
 [: "\qh " (Ctrl-[, Question mark over hat)  
 ]: "\hd " (Ctrl-], Hat over, dot under)

**Table 2. Table showing how diacritical marks appear, are produced and the commands that appear on the screen**

	VOWEL	GRAVE	ACUTE	TILDE	QUEST.	DOT
	A	À	Á	Ã	Â	Ạ
	a	à	á	ã	â	ạ
TYPE		ctl-z	ctl-x	ctl-v	ctl-b	ctl-n
SCREEN		\g	\a	\t	\q	\d
	Â	À	Á	Ã	Â	Ạ
	â	à	á	ã	â	ạ
TYPE	ctl-u	ctl-i	ctl-o	ctl-p	ctl-[	ctl-]
SCREEN	\h	\gh	\ah	\th	\qh	\hd
	Ă	À	Á	Ã	Â	Ạ
	ă	à	á	ã	â	ạ
TYPE	ctl-j	ctl-k	ctl-l	ctl-m	shift-2	shift-3
SCREEN	\u	\gu	\au	\tu	\qu	\ud
	E	È	É	Ë	Ê	Ệ
	e	è	é	ë	ê	ệ
TYPE		ctl-z	ctl-x	ctl-v	ctl-b	ctl-n
SCREEN		\g	\a	\t	\q	\d
	Ê	È	É	Ë	Ê	Ệ
	ê	è	é	ë	ê	ệ
TYPE	ctl-u	ctl-i	ctl-o	ctl-p	ctl-[	ctl-]
SCREEN	\h	\gh	\ah	\th	\qh	\hd
	I	Ì	Í	Ĩ	Î	Ị
	i	ì	í	ĩ	î	ị
TYPE		ctl-z	ctl-x	ctl-v	ctl-b	ctl-n
SCREEN		\g	\a	\t	\q	\d

	Ō	Ò	Ó	Õ	Ȫ	ȫ
	o	ò	ó	õ	ȫ	ȫ
TYPE SCREEN		ctl-z \g	ctl-x \a	ctl-v \t	ctl-b \q	ctl-n \d
	Ô	Ë	Ó	Õ	Ȫ	ȫ
	ô	ë	ó	õ	ȫ	ȫ
TYPE SCREEN	ctl-u \h	ctl-i \gh	ctl-o \ah	ctl-p \th	ctl-[ \qh	ctl-] \hd
	Ū	Û	Ū	Ū	Ū	Ū
	u	ù	ú	ũ	ű	ü
TYPE SCREEN		ctl-z \g	ctl-x \a	ctl-v \t	ctl-b \q	ctl-n \d
	Ų	Ų	Ų	Ų	Ų	Ų
	Ų	Ų	Ų	Ų	Ų	Ų
TYPE SCREEN	ctl-q \ob	ctl-w * \bg o\s	ctl-e * \ba o\s	ctl-r * \bt o\s	ctl-t * \bq o\s	ctl-y * \bd o\s
	Ŵ	Ŵ	Ŵ	Ŵ	Ŵ	Ŵ
	Ŵ	Ŵ	Ŵ	Ŵ	Ŵ	Ŵ
TYPE SCREEN	ctl-a \ub	ctl-s * \bg u\s	ctl-d * \ba u\s	ctl-f * \bt u\s	ctl-g * \bq u\s	ctl-h * \bd u\s
	Ŷ	Ŷ	Ŷ	Ŷ	Ŷ	Ŷ
	Ŷ	Ŷ	Ŷ	Ŷ	Ŷ	Ŷ
TYPE SCREEN	ctl-c q * \cob	ctl-c w * \cobg	ctl-c e * \coba	ctl-c r * \cobt	ctl-c t * \cobq	ctl-c y * \codb
	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
TYPE SCREEN	ctl-c a * \cub	ctl-c s * \cubg	ctl-c d * \cuba	ctl-c f * \cubt	ctl-c g * \cubq	ctl-c h * \cubd

\* indicates that the control sequence introduces its own letter and hence the letter present in the text must be deleted.

Table 3. Macros for definitions used. Found in file: vdefq.

```

\font\vf=font
\def\g{\'}
\def\af{\'}
\def\t{\~}
\def\h{\^}
\def\au{\leavevmode\raise.929ex\rlap{\kern.1em\accent"13 }%
  \raise.465ex\rlap{\kern.1em\accent"15 }}
\def\hd{\leavevmode\raise.465ex\rlap{\kern.1em\accent"5E }%
  \lower.465ex\rlap{\kern.2em.}}
\def\q{\leavevmode\raise1.394ex\rlap{\kern.2em\v ?}}
\def\gu{\leavevmode\raise.929ex\rlap{\kern.1em\accent"12 }%
  \raise.465ex\rlap{\kern.1em\accent"15 }}
\def\tu{\leavevmode\raise1.161ex\rlap{\kern.1em\accent"7E }%
  \raise.465ex\rlap{\kern.1em\accent"15 }}
\def\qu{\leavevmode\raise1.858ex\rlap{\kern.2em\v ?}%
  \raise.465ex\rlap{\kern.1em\accent"15 }}
\def\ud{\leavevmode\raise.465ex\rlap{\kern.1em\accent"15 }%
  \lower.465ex\rlap{\kern.15em.}}
\def\gh{\leavevmode\raise1.161ex\rlap{\kern.15em\accent"12 }%
  \raise.465ex\rlap{\kern.1em\accent"5E }}
\def\ah{\leavevmode\raise1.161ex\rlap{\kern.05em\accent"13 }%
  \raise.465ex\rlap{\kern.1em\accent"5E }}
\def\th{\leavevmode\raise1.161ex\rlap{\kern.1em\accent"7E }%
  \raise.465ex\rlap{\kern.1em\accent"5E }}
\def\qh{\leavevmode\raise2.090ex\rlap{\kern.2em\v ?}%
  \raise.465ex\rlap{\kern.1em\accent"5E }}
\def\ob{o\lower.604ex\hbox{\kern-.25em\accent"15 }} % o with a beard
\def\ub{u\lower.604ex\hbox{\kern-.25em\accent"15 }} % u with a beard
\def\s{\hbox{\kern.2em}} % adds two pts space
\def\bg#1{\'#1\b}
\def\ba#1{\'#1\b}
\def\bt#1{\~#1\b}
\def\bq#1{q#1\b}
\def\bd#1{d#1\b}
\def\b{\lower.604ex\hbox{\kern-.25em\accent"15 }}
\def\cob{O\b} % O with a beard
\def\cub{U\b} % U with a beard
\def\cobg{\leavevmode\raise.698ex\rlap{\kern.2em\accent"12 }\cob}
  % O-beard and accent grave
\def\cobaf{\leavevmode\raise.698ex\rlap{\kern.2em\accent"13 }\cob}
  % O-beard and accent acute
\def\cobt{\leavevmode\raise.698ex\rlap{\kern.2em\accent"7E }\cob}
  % O-beard and title
\def\cobd{\leavevmode\lower.465ex\rlap{\kern.2em.}\cob}
  % O-beard and dot below
\def\cobq{\leavevmode\raise1.626ex\rlap{\kern.3em\v ?}\cob}
  % O-beard and question mark
\def\cubg{\leavevmode\raise.698ex\rlap{\kern.2em\accent"12 }\cub}
  % U-beard and accent grave
\def\cubaf{\leavevmode\raise.698ex\rlap{\kern.2em\accent"13 }\cub}
  % U-beard and accent acute
\def\cubt{\leavevmode\raise.465ex\rlap{\kern.15em\accent"7E }\cub}
  % U-beard and title
\def\cubq{\leavevmode\raise1.626ex\rlap{\kern.25em\v ?}\cub}
  % U-beard and question mark
\def\cubd{\leavevmode\lower.465ex\rlap{\kern.2em.}\cub}
  % U-beard and dot below

```

---

## A new font selection scheme for T<sub>E</sub>X macro packages — the basic macros

Frank Mittelbach  
Rainer Schöpf  
Johannes Gutenberg Universität Mainz

### Abstract

We have implemented a new font selection scheme for T<sub>E</sub>X and its macro packages. This scheme allows font family, series, shape, and size to be specified independently. Additionally, it is not necessary to preload all math fonts.

---

### Contents

<b>1 Introduction</b>	<b>222</b>	4.1 Handling the font tables . . . . .	225
<b>2 The User Interface</b>	<b>222</b>	4.2 Fonts for math . . . . .	225
2.1 Selection of a new font . . . . .	223	4.3 Special considerations . . . . .	227
2.2 Changing the math <i>version</i> . . . . .	223	<b>5 Preliminary macros</b>	<b>227</b>
<b>3 Setting up a new format</b>	<b>223</b>	<b>6 Macros for setting up the tables</b>	<b>228</b>
3.1 Defining a new <i>family/series/shape</i> combination . . . . .	223	<b>7 Selecting a new font</b>	<b>229</b>
3.2 Preloading fonts . . . . .	224	7.1 Macros for the user . . . . .	229
3.3 Defining math <i>groups</i> . . . . .	224	7.2 Macros for loading fonts . . . . .	230
<b>4 Concept of the implementation</b>	<b>225</b>	<b>8 Assigning math fonts to <i>versions</i></b>	<b>234</b>

---

## 1 Introduction

In traditional typesetting one distinguishes four parameters to describe a font: the font *family* (e.g. computer modern), the font *series* (e.g. roman or sansserif), the font *shape* (e.g. normal or bold), and the font *size*. This distinction is not always unique: take for example the slanted typeface L<sup>A</sup>T<sub>E</sub>X uses. This can be seen as the sloped *shape* of *series* roman or as the normal *shape* of *series* sloped.

Recently several people have asked how to use such a scheme in L<sup>A</sup>T<sub>E</sub>X. Unfortunately the current implementation of L<sup>A</sup>T<sub>E</sub>X's font selection scheme does not allow incorporation of this concept.

When typesetting math formulas, one usually needs many more fonts than for ordinary text. In the T<sub>E</sub>Xbook Donald Knuth says:

All characters that are typeset in math mode belong to one of sixteen *families of fonts*<sup>1</sup>, numbered internally from 0 to 15.

The use of the word *family* in this context is unfortunate; it conflicts with the font families we are talking about. To avoid confusion we will always speak

of font families from the typesetter's point of view. For math we speak of *math groups* each connected to three fonts called the `\textfont`, the `\scriptfont`, and the `\scriptscriptfont`. From the user's point of view, math formulas consist of characters coming from specific math *alphabets* (e.g. those selected by `\cal`) and of symbols (e.g. `\sum`) selected by a special control sequence and scattered over a number of fonts.

All fonts that can be used together in *one* math formula form a *version*. *Versions* can only be switched outside math formulas. Standard L<sup>A</sup>T<sub>E</sub>X provides two *versions*: normal and bold.

## 2 The User Interface

The commands described in the next subsections are primitives used to build up more powerful interfaces. But they are all user accessible. We used these commands to construct two interfaces for L<sup>A</sup>T<sub>E</sub>X: one is mimicking the old font selection (e.g. `\bf` is used to switch to the font `cmbx..`), the other one implements an orthogonal font selection scheme (here

---

<sup>1</sup> Emphasis by DEK



`\bf` means: change the current *shape* and select a new font but leave *family*, *series* and *size* untouched.) Details can be found in the article "The new font family selection — User Interface to standard L<sup>A</sup>T<sub>E</sub>X".

## 2.1 Selection of a new font

Selecting a new font is done in two independent steps. First you have to change the values for *family*, *series*, *shape* and/or *size* and then execute a macro which uses the new values to select the desired font. If you don't use this macro the font will not be changed.

The first step is done with the macros `\family`, `\series`, `\shape` and `\size`. For example, if you want to switch to the 'sansserif' *series* you have to say `\series{sansserif}`. Except for `\size`, all those macros have one argument, namely, the desired *family*, *series* or *shape*, respectively. The macro `\size` is somewhat special because we decided that it would be better to force the user to specify a new *size* and a `\baselineskip` together for this *size*, so the macro has two arguments.

All four macros will silently accept their arguments. Warning messages are generated in the second step when the actual font selection is carried out.

To select a (new) font one has to call the `\selectfont` macro. This macro looks up the current *family*, *series*, *shape* and *size*, possibly changed by one of the above mentioned commands, and switches to this font, provided the selected combination of *family*, *series* and *shape* is known to the system. If it is unknown, a warning will be printed, and up to three new trials are made to find a substitute. This is done by changing to `\default@shape`, then to `\default@series` and as a last resort to `\default@family`. At least this combination must have been defined, otherwise we will find ourselves in an endless loop.<sup>2</sup>

It may still be, however, that the *size* requested is not specified in the table. This will lead to an error, and the font given by `\default@errfont` will be selected. All four defaults are given private names (names containing an `@`) to emphasize that their values should be changed only by the "local wizards".

The selection scheme described above may seem unnecessarily complicated. But consider the follow-

ing example: you are now reading a sentence typeset in the `cmr10` font, that is *family* 'computer modern', *series* 'roman', *shape* 'normal' and *size* '10'. If we want to switch to 'typewriter italic' we say `\series{typewriter} \shape{italic}` and then `\selectfont`. To avoid the call to `\selectfont` we would have to embed it in the definition of `\series`, etc. But this means that either `cmr10` or `cmr10` would be unnecessarily selected (and probably loaded).

As mentioned before, these commands are primitive; they should be used to define higher level commands for a special application. For example L<sup>A</sup>T<sub>E</sub>X's `\bf` command can be defined as

```
\def\bf{\series{roman}%
\shape{boldext}%
\selectfont}
```

to work in the same way as before in L<sup>A</sup>T<sub>E</sub>X.<sup>3</sup> As an alternative, the definition might be

```
\def\bf{\shape{boldext}\selectfont}
```

which will change to the 'bold extended' *shape* in the current *family*, *series* and *size*.

## 2.2 Changing the math version

`\mathversion` switches to another math *version*, e.g.,

```
\mathversion{bold}
```

will switch to *version* 'bold' provided that it is defined. This command can be used only outside of math formulas. As an example we give the definitions of Standard L<sup>A</sup>T<sub>E</sub>X's `\boldmath` and `\unboldmath` macros in terms of `\mathversion`. For this we must assume that two *versions* 'cmnormal' and 'cmbold' are already defined.<sup>4</sup>

```
\def\boldmath{\@nomath\boldmath
\mathversion{cmbold}}
\def\unboldmath{\@nomath\unboldmath
\mathversion{cmnormal}}
```

## 3 Setting up a new format

### 3.1 Defining a new family/series/shape combination

Assume that you want to define the combination *family* 'computer modern', *series* 'concrete', *shape* 'italic'. In the present case we have to write

```
\new@fontshape{cm}{concrete}{italic}{%
<5>1ccr5%
```

<sup>2</sup> This can be fixed easily but we are not sure if it's worth the effort. The defaults shouldn't be changed by an ordinary user job, and it's not necessary to provide code to check a format file.

<sup>3</sup> Actually this definition behaves differently when used in math mode, because then no **bold** face is selected. We will see a correct definition later.

<sup>4</sup> The `\@nomath` command used here issues a warning if these commands are used in math mode.

```

<6>1ccr6%
<7>1ccr7%
<8>ccti10 at 8pt%
<9>ccti10 at 9pt%
<10>ccti10%
<11>ccti10 at 10.95pt%
<12>ccti10 at 12pt%
<14>ccti10 at 14.4pt%
<17>ccti10 at 17.28pt%
<20>ccti10 at 20.74pt%
<25>ccti10 at 24.88pt}

```

The general form of the specification in the fourth argument of `\new@fontshape` is

```
<(size)>(external font name)
```

You are totally free in what you write between the `<>` to denote the size.

If you look closely at the example given above you'll notice that the first three lines (for sizes 5, 6, and 7) seem to be wrong: they start with a 1 and the external font names are incorrect. This is a special feature of the font selection code that allows font substitution. The numbers in front of the external font name mean:

- 0 No effect. Same as no number at all.
- 1 Issue a warning that the requested *family/series/shape* combination is not available in this size and use the font given instead.
- 2 Issue a warning that the requested *family/series* does not contain the requested *shape* and use the font given instead.

Additionally, for every *family/series* combination there exists a so-called 'extra' macro that is used to set parameters, etc. common to all *shapes* and *sizes*, e.g. inhibiting hyphenation for typewriter fonts. Its argument is the internal font name.

```

\extra@def{cm}{typewriter}%
    {hyphenchar#1\m@ne}

```

### 3.2 Preloading fonts

The macro `\preload@sizes` provides an easy way to specify fonts that should be preloaded when dumping a format file. It is used as follows:

```

\preload@sizes{(family)}{(series)}{(shape)}
    {(list of sizes)}

```

where the elements of *(list of sizes)* are delimited by commas. Note that it makes no difference for your documents whether you preload a font or load it on demand. In the latter case, however, processing documents takes more time.

### 3.3 Defining math groups

To specify fonts for math, other primitive commands are provided. They all have `@` characters in their

names; i.e. they will not normally be accessible to the user, but will be when making format files or style files.

Math fonts can be divided in two classes: fonts that are accessed via `\mathchardef` and those that are selected *only* via a *(math alphabet identifier)*.

As we already mentioned, all math fonts come in *groups* of `\textfont`, `\scriptfont`, and `\scriptscriptfont`. A new math *group* is defined by the command

```
\new@mathgroup(math group number)
```

*(math group number)* is a control sequence that is assigned a number that from now on will denote this *group*. It is also possible to use an explicit number, i.e. a sequence of digits, instead of this control sequence to stand for this *group*. However, the first alternative is generally superior since `\new@mathgroup` always assigns a previously unused number to this control sequence. The second alternative is normally used for *groups* 0, 1, 2, and 3 which have a special meaning to  $\TeX$ .

To specify the fonts of this *group*, the commands `\define@mathgroup` (for the first class) and `\define@mathalphabet` (for the second class) are available.

Take for example one of the `cmsy..` fonts, i.e. the standard math symbol fonts in the computer modern family. (They also contain the calligraphic alphabet.) This font must be loaded prior to its use because of the `\mathchardef` commands in `plain.tex`. To achieve this we write

```

\define@mathgroup {cmnormal}2%
    {cm}{mathsymbol}{normal}

```

This can be read as: define the *group* number 2 in the 'cmnormal' *version* to consist of fonts with *family* 'cm', *series* 'mathsymbol', and *shape* normal. The actual *sizes* for `\textfont`, `\scriptfont`, and `\scriptscriptfont` will be determined when the *group* is selected.

If you want to access such a math *group* also via a *(math alphabet identifier)* you must define this control sequence to switch to the corresponding internal *group* *(number)*, viz.

```

\def(math alphabet identifier){%
    \group(number)}

```

Returning to our example: to define `\cal` to select the calligraphic alphabet ( $\mathcal{A}$ ,  $\mathcal{B}$ ) in a formula one has to add the definition

```
\def\cal{\group2 }
```

If we want to declare a *group* that is accessed always by a *(math alphabet identifier)* the `\define@mathalphabet` macro should be used. Since the corresponding fonts are not accessed

by `\mathchardef` commands, there is no need to preload them. Loading can be done by the `\math alphabet identifier`.

The macro `\define@mathalphabet` is similar to `\define@mathgroup`. If one uses a macro `\sfmath` to select sansserif letters in a formula one has to make a declaration like

```
\define@mathalphabet{cmnormal}\sfmath
  {math group number}{cm}{sansserif}{normal}
```

Here `\sfmath` is the new `\math alphabet identifier`. The `\math group number` that must previously be defined using `\new@mathgroup`.

The text size in math formulas is always determined to be the *size* of the text outside. The sizes for subscripts, etc., i.e. the script and the scriptscript size, must be specified additionally.

The macro `\define@mathsizes` is made for this purpose. It takes three arguments: a text size, the corresponding script size and scriptscript size, e.g.

```
\define@mathsizes{10}{7}{5}
```

defines the script and scriptscript sizes for a text set in *size* '10' to be '7' and '5', resp.

When a *size* change occurs, not only the current font must be switched but also all math fonts which can be selected via special symbols. On the other hand, it may be that math fonts are only available or only used in certain *sizes*. For the other *sizes* we do not need to switch the whole set of math fonts.<sup>5</sup> To specify this we provide the command `\define@nomathsize` that takes only the text size and inhibits math font switching for this *size*.

If there is more than one *version* provided then you better define all *groups* for every *version*. Otherwise switching the *version* (by using `\mathversion`) will not reset these *groups* properly. E.g., in the L<sup>A</sup>T<sub>E</sub>X implementation we therefore have a line

```
\define@mathgroup{cmbold}2%
  {cm}{mathsymbol}{bold}
```

It goes without saying that the *family/series/shape* combination must have been defined previously by a `\new@fontshape` command.

## 4 Concept of the implementation

### 4.1 Handling the font tables

The first problem we had to solve was how to handle such a huge number of fonts. To implement the four dimensional grid of fonts we maintain an association list<sup>6</sup> (i.e. a list of pairs) with elements (*size*, *external font name*) for every combination of font *family/series/shape*. We do not redefine the font chang-

ing commands: these commands select the correct font by looking into the association list corresponding to the current font *family/series/shape* combination. This association list is hidden in a macro. Its precise form is as follows: For every `\size` we have a string of the form

```
<\size>\external font name
```

This strings are simply concatenated to form one long string of characters. In this way all necessary information is available. But this solution would take up far too much of T<sub>E</sub>X's valuable main memory. Therefore we use a trick, the same trick that is used in plain T<sub>E</sub>X's `\newhelp` macro: we enclose the list of characters by `\csname... \endcsname` making one macro name out of it. This uses up only one token in T<sub>E</sub>X's main memory (and some string memory but this is comparatively cheap).

As an example take the normal *shape* of *series* roman in the computer modern *family*, i.e. the cmr fonts. We define a macro `\cm/roman/normal` whose replacement text contains a single token containing all necessary information in its name. This macro itself is undefined.

```
\expandafter\edef
  \csname cm/roman/normal\endcsname{%
  \expandafter\noexpand\csname
  <V>cmr5%
  <VI>cmr6%
  <VII>cmr7%
  <VIII>cmr8%
  <IX>cmr9%
  <X>cmr10%
  <XI>cmr10 at 10.95pt%
  <XII>cmr12%
  <XIV>cmr12 at 14.4pt%
  <XVII>cmr17%
  <XX>cmr17 at 20.736pt%
  <XXV>cmr17 at 24.8832pt%
  \endcsname}
```

The first `\expandafter` is needed because the macro name consists of / characters and we use `\csname... \endcsname` to build them into the macro name. We then use `\edef` so that the second `\csname... \endcsname` combination is expanded at definition time. Finally we need the `\expandafter\noexpand` trick to ensure that the resulting (undefined) macro is not expanded.

### 4.2 Fonts for math

To set up fonts for math one has to set up several assignments of the form

<sup>5</sup> Think of a special *size* used only in titles with no formulas at all.

<sup>6</sup> Lisp hackers note!

$\langle math font assignment \rangle \langle number \rangle = \langle font \rangle$

where  $\langle math font assignment \rangle$  is one of `\textfont`, `\scriptfont`, or `\scriptscriptfont`,  $\langle number \rangle$  is a number associated with the particular font *group* (or family in the terminology used by DEK), and  $\langle font \rangle$  the internal name of the font to be selected. The assignments have to be changed when the overall size changes or when a new math *version* is requested by the user.

There are several ways to implement this: one can for example build a macro name from the requested *version* and the current size (i.e. `\f@size`). The replacement text would then contain all necessary assignments. In a way the old font selection scheme of L<sup>A</sup>T<sub>E</sub>X uses this method by defining macros like `\xpt`, etc.

We decided to use another approach: Since the current size (`\f@size`) is always known we make all necessary assignments by means of one macro to be called for every *version*, viz.

`\getanddefine@fonts\langle number \rangle \langle font shape \rangle`,

where  $\langle number \rangle$  has the same meaning as before, and  $\langle font shape \rangle$  denotes a macro name like `\cm/mathsymbol/normal` which can be used to get the necessary fonts names by appending the desired size. For every text size there exists a script and a scriptscript size. Our method for getting it will be seen in a minute (depending on your speed of reading).

On first sight this seems to be a lot slower than the other method because `\getanddefine@fonts` takes time to put all the information together.<sup>7</sup> But tests have shown that this is not true: we can neglect this extra time.

The above math font assignments must be done for all fonts containing characters accessed via `\mathchardef` because T<sub>E</sub>X will complain if these fonts are not properly defined when these symbols are used. (The alternative to convert all `\mathchardef`'s into macro calls that test if the font is available seems to be too inefficient but this should be investigated further.)

For math *alphabets* the situation is different. These are selected by means of the corresponding  $\langle math alphabet identifier \rangle$  (i.e. `\cal`) so that the fonts can be loaded on demand.<sup>8</sup> Hence in the *version* macros, for these font groups we have lines of the form

`\def\langle math alphabet identifier \rangle\{%`  
`\select@group\langle math alphabet identifier \rangle`

<sup>7</sup> The old song: Time vs. space!

<sup>8</sup> However, the example of `\cal` is a bad one because the fonts containing this *alphabet* must be preloaded anyway. Those fonts also contain the bulk of math symbols accessed via `\mathchardef`.

$\langle number \rangle \langle font shape \rangle \}%$

$\langle number \rangle$  and  $\langle font shape \rangle$  have the same meaning as before;  $\langle math alphabet identifier \rangle$  is available to the user to select the math *alphabet*. The actual math font assignments are carried out by the macro `\select@group` which is called only if the user selects the *alphabet* inside a formula. In this way, fonts not used in a certain document are not loaded, thereby saving space and time.

For every math *alphabet*, there must obviously exist at least one *version*. But it is perfectly legal that certain *alphabets* are available only in certain *versions*. Therefore we need a way to warn the user if he selects a *version* of an *alphabet* that does not exist.

If a *math alphabet* does not exist in a certain *version* the corresponding part of the *version* macro will look like

`\def\langle math alphabet identifier \rangle\{%`  
`\no@version@warning\langle version \rangle`  
`\langle math alphabet identifier \rangle \}%`

which leads to a warning message if the *alphabet* is selected in this *version*.

A minute ago we promised to tell how we obtain the script and scriptscript sizes for a given text size. For every size and math *group*, you need a `\textfont`, a `\scriptfont`, and a `\scriptscriptfont`. The math fonts have to be switched for every size change. Since the math *group* assignments have to be in effect when the current math formula ends we make them all global. But then the old assignments must be restored at the end of the current group. This is done by inserting a macro call with the `\aftergroup` primitive. The current text size is always available to this macro in the expansion of `\f@size`. The corresponding script size and scriptscriptsize (specified via `\define@mathsizes`), however, must be recorded somewhere. We use the following scheme for this: for every size *s* we define a macro `\S@s` (e.g. for size *XX* we define `\S@XX`) that globally defines two macros `\sf@size` and `\ssf@size` to expand to the corresponding script size and scriptscript size. With the help of these macros the right sizes can be extracted easily. Take for example size '10', with script size '7' and scriptscript size '5'. The corresponding macro looks like

`\expandafter\def\csname S@10\endcsname`  
`{\gdef\sf@size{7}\gdef\ssf@size{5}}`

### 4.3 Special considerations

There are two special cases we must take care of. Both have to do with size changes within an alignment. Why is this special? The first problem appears when the size change occurs in the last column of an alignment. The token saved by the `\aftergroup` primitive will be inserted just after the `\cr` has been read. More

precisely: after the end of the alignment template. But here only `\noalign` or the end of the alignment is allowed, everything else starts a new column. There is a simple fix for this: in the template of the alignment the hash mark (#) denoting the last column must be wrapped in a group. The same problem shows up if a size change occurs inside a `\noalign`.

## 5 Preliminary macros

As always we begin by identifying the latest version of this file on the VDU and in the log file.

```
\immediate\write\sixt@@n{File: 'fam.tex'
      \fileversion \space <\filedate> (FMI and RmS)}
\immediate\write\sixt@@n{English Documentation
      <\docdate> (RmS and FMI)}
```

Following are a number of macros that will be used later.

`\@spaces` We define `\@spaces` to be an abbreviation for five space tokens.

```
\def\@spaces{\space\space\space\space\space}
```

This is also defined in `latex.tex`, but this code cautiously does not assume that any macros are defined elsewhere (except those in `plain.tex`).

`\@gobble` The `\@gobble` macro is used to get rid of its argument.

```
\def\@gobble#1{}
```

`\@empty` The `\@empty` macro expands to nothing and is used to test for empty replacement texts.

```
\def\@empty{}
```

`\@height`, `\@depth` and `\@width` The `\@height`, `\@depth` and `\@width` macros are made to conserve token memory.

```
\@height \def\@height{height}
```

```
\@depth \def\@depth{depth}
```

```
\@width \def\@width{width}
```

`\font@warning` We need a macro that prints a warning message. We write to output stream 16 which means that the message will appear both in the transcript file and on the terminal.

```
\def\font@warning#1{\immediate \write \sixt@@n {Warning: #1.}}
```

`\@nomath` `\@nomath` is used by all macros that should not be used in math mode.

```
\def\@nomath#1{\relax\ifmmode \font@warning{Don't use \string#1 in
      math mode}\fi}
```

`\no@version@warning` The macro `\no@version@warning` is called whenever the user requests a math *alphabet* that is not available in the current *version*. The first argument is the name of the *version* (as a sequence of characters), the second is the control sequence that identifies the math *alphabet*. The `\relax` at the beginning is necessary to prevent `TEX` from scanning too far in certain situations.

```
\def\no@version@warning#1#2{\relax \ifmmode
      \font@warning{No '#1' version for math alphabet identifier
      \string#2}\fi}
```

`\new@mathgroup` We have to redefine one plain T<sub>E</sub>X macro: We must remove `\outer` from definition of `\newfam` so that it can be used inside other macros. We also give a new name to `\newfam` and `\fam` to avoid verbal confusion (see the introduction).<sup>9</sup>

```
\def\new@mathgroup{\alloc@8\group\chardef\sixt@@n}
\let\group\fam
%\let\newfam\relax
%\let\fam\relax
```

## 6 Macros for setting up the tables

`\new@fontshape` Since this kind of definition is needed several times we provide a macro `\new@fontshape` that does the work for us.

```
\def\new@fontshape#1#2#3#4{\expandafter
\edef\csname#1/#2/#3\endcsname{\expandafter\noexpand
\csname #4\endcsname}}
```

`\extra@def` The 'extra' macro is defined as follows.

```
\expandafter\def\csname extra//cm/typewriter\endcsname#1{%
\hyphenchar#1\m@ne}
```

We provide an abbreviation for this:

```
\def\extra@def#1#2#3{%
\expandafter\def\csname extra//#1/#2\endcsname##1{#3}}
```

so that the above definition looks like

```
\extra@def{cm}{typewriter}{\hyphenchar#1\m@ne}
```

However, this is inefficient if there is nothing to do (i.e. if the third argument is empty), so we provide a special test for this case. Here is the actual definition:

```
\def\extra@def#1#2#3{%
```

We store the argument #3 in a temporary macro `\@tempa`. This must have one parameter since #1 is allowed in the third argument of `\extra@def` (otherwise T<sub>E</sub>X will not accept the definition).

```
\def\@tempa##1{#3}%
```

We compare `\@tempa` with a macro with one argument and empty replacement text, i.e. with `\@gobble`. If these two are the same, we `\let` the 'extra' macro equal `\@gobble`.

```
\ifx \@tempa\@gobble
\expandafter\let\csname extra//#1/#2\endcsname\@gobble
```

Otherwise, we build a definition.

```
\else \expandafter\def\csname extra//#1/#2\endcsname##1{#3}\fi}
```

`\preload@sizes` As we already explained, the macro `\preload@sizes` provides a convenient way to specify fonts to be preloaded. It takes four arguments and its definition is as follows:

```
\def\preload@sizes#1#2#3#4{%
```

We define a macro `\next`<sup>10</sup> that grabs the next *size* and loads the corresponding font. This is done by delimiting `\next`'s only argument by the token , (comma).

```
\def\next##1,{%
```

<sup>9</sup> For the same reason it seems advisable to `\let\fam` and `\newfam` equal to `\relax`, but this is commented out to retain compatibility to existing style files.

<sup>10</sup> We cannot use `\@tempa` since it is needed in `\pickup@font`.

The end of the list will be detected when there are no more elements, i.e. when `\next's` argument is empty. The trick used here is explained in Appendix D of the `TEXbook`: if the argument is empty, the `\if` will select the first clause and `\let \next` equal to `\relax`. (We use the `>` character here since it cannot appear in font file names.)

```
\if>##1>
  \let\next\relax
\else
```

Otherwise, we define `\font@name` appropriately and call `\pickup@font` to do the work. Note that the requested *family/series/shape* combination must have been defined, or you will get an error.

```
\edef\font@name{\csname#1/#2/#3/##1\endcsname}%
\pickup@font
\fi
```

Finally we call `\next` again to process the next *size*. If `\next` was `\let` equal to `\relax` this will end the macro.

```
\next}%
```

We finish by reinserting the list of sizes after the `\next` macro and appending an empty element so that the end of the list is recognized properly.

```
\next#4,,}
```

`\ifdefine@mathfonts` We need a switch to decide if we have to change math fonts. For this purpose we provide `\ifdefine@mathfonts` that can be set to true or false by the `\S@...` macros, depending on whether math fonts are provided for this size or not. The default is, of course, to switch all fonts.

```
\newif\ifdefine@mathfonts \define@mathfontstrue
```

`\define@mathsizes` `\define@mathsizes` takes the text size, script size, and scriptscript size as arguments and defines the right `\S@...` macro. (`\define@mathfontstrue` might be omitted if math fonts are to be defined for every size.)

```
\def\define@mathsizes#1#2#3{\expandafter \def
  \csname S@#1\endcsname{\gdef\sf@size{#2}\gdef\ssf@size{#3}%
  \define@mathfontstrue}}
```

`\define@nomathsize` `\define@nomathsize` takes only the text size as argument and defines `\S@...` to not change math fonts.

```
\def\define@nomathsize#1{\expandafter \let
  \csname S@#1\endcsname \define@mathfontsfalse}
```

## 7 Selecting a new font

### 7.1 Macros for the user

`\family` As we said in the introduction, a font is described by four parameters. We first define `\series` macros to specify the desired *family*, *series*, or *shape*. These are simply recorded in internal macros `\f@family`, `\f@series`, and `\f@shape`, resp. We use `\edef's` so that `\f@family` the arguments can also be macros.

```
\f@family \def\family#1{\edef\f@family{#1}}
\f@series \def\series#1{\edef\f@series{#1}}
\f@shape \def\shape#1{\edef\f@shape{#1}}
```

`\size` We also define a macro that allows specification of a size. In this case, however, we `\f@size` also need the value of `\baselineskip`. We cannot set `\baselineskip` immediately, so it is recorded in the macro `\setnew@baselineskip`. We use `\edef` here because the second argument (`#2`) might be a macro.

```
\def\size#1#2{%
  \edef\f@size{#1}%
  \edef\setnew@baselineskip{\baselineskip #2\relax}}
```

`\selectfont` The macro `\selectfont` is called whenever a font change must take place.

```
\glb@currsize \def\selectfont{%
  Its first action is to determine if the new font has the same size as the previous one.
  Here the macro \glb@currsize holds the current font size. Its expansion text may
  also be empty which means that we do not know what the current size is. As its name
  indicates, it is always set globally.
  \ifx \glb@currsize \f@size
  If the size is to be changed we must also change \baselineskip and a number of
  other parameters. This is done by the macro \glb@settings.
  \else \glb@settings
  Since these changes are done globally, we must ensure that the old values are re-
  stored at the end of the current group. We use TeX's \aftergroup primitive to call
  \glb@settings again just after the current group ends. And that's all of special code
  for a size change.
  \aftergroup\glb@settings \fi
  We now generate the internal name of the font by concatenating family, series, shape,
  and current size, with slashes as delimiters between them. This is much more readable
  than standard LATEX's \twfbf, etc.
  \edef\font@name{%
    \csname \f@family/\f@series/\f@shape/\f@size\endcsname}%
  We call the macro \pickup@font which will load the font if necessary.
  \pickup@font
  Finally, we select the font. This finishes the macro \selectfont.
  \font@name}
```

`\mathversion` `\math@version` `\mathversion` takes the *math version* name as argument, defines `\math@version` appropriately and switches to the font selected, forcing a call to `\glb@settings` if the *version* is known to the system.

```
\def\mathversion#1{\expandafter\ifx\csname #1\endcsname\relax
  \font@warning{The requested version '#1' is unknown}\else
  \def\math@version{#1}\glb@settings\aftergroup\glb@settings\fi}
```

## 7.2 Macros for loading fonts

`\pickup@font` The macro `\pickup@font` which is used in `\selectfont` is very simple: if the font name is undefined (i.e. not known yet) it calls `\define@newfont` to load it.

```
\def\pickup@font{%
  \expandafter \ifx \font@name \relax
  \define@newfont
  \fi}
```

`\split@name` `\pickup@font` assumes that `\font@name` is set but it is sometimes called when `\f@family`, `\f@series`, `\f@shape`, or `\f@size` may have the wrong settings (see, e.g., the definition of `\getanddefine@fonts`). Therefore we need a macro to extract font *family*, *series*, *shape*, and *size* from the font name. To this end we define `\split@name` which takes the font name as a list of characters of `\catcode 12` (without the backslash at the beginning) delimited by the special control sequence `\@nil`. This is not very complicated: we first ensure that `/` has the right `\catcode`

```
{\catcode'\/=12
```

and define `\split@name` so that it will define our private `\f@family`, `\f@series`, `\f@shape`, and `\f@size` macros.

```
\gdef\split@name#1/#2/#3/#4\@nil{\def\f@family{#1}%
```



```

\def\f@series{#2}%
\def\f@shape{#3}%
\def\f@size{#4}}

```

`\define@newfont` Now we can tackle the problem of defining a new font.

```

\def\define@newfont{%

```

We have already mentioned that the token list that `\split@name` will get as argument must not start with a backslash. To reach this goal, we will set the `\escapechar` to -1 so that the `\string` primitive will not generate an escape character. But then we must save `\escapechar`'s current value. We use count register `\count@` for this purpose.

```

\count@\escapechar
\escapechar\m@ne

```

Then we extract *family*, *series*, *shape*, and *size* from the font name. Note the four `\expandafte`rs so that `\font@name` is expanded first, then `\string`, and finally `\split@name`.

```

\expandafter\expandafter\expandafter
\split@name\expandafter\string\font@name\@nil

```

If the *family/series/shape* combination is not available (i.e. undefined), we call the macro `\wrong@fontshape` to take care of this case. Otherwise, `\extract@font` will load the external font for us.

```

\expandafter\ifx
\csname\f@family/\f@series/\f@shape\endcsname \relax
\wrong@fontshape\else \extract@font\fi

```

We are nearly finished and must only restore the `\escapechar`.

```

\escapechar\count@}

```

`\wrong@fontshape` Before we come to the macro `\extract@font`, we have to take care of unknown *family/series/shape* combinations. The general strategy is to issue a warning and to try a default *shape*, then a default *series*, and finally a default *family*. If this last one also fails, T<sub>E</sub>X will go into an infinite loop. But if the defaults are incorrectly set, one deserves nothing else!

```

\def\wrong@fontshape{%

```

We remember the desired *family/series/shape* combination which we will need in a moment.

```

\edef\@tempa{\csname\f@family/\f@series/\f@shape\endcsname}%

```

Then we warn the user about the mess and set the shape to its default.

```

\font@warning{Font/shape '\@tempa' unknown}%
\shape\default@shape

```

If the combination is not known, try the default *series*.

```

\expandafter\ifx\csname\f@family/\f@series/\f@shape\endcsname\relax
\series\default@series

```

If this is still undefined, try the default *family*. Otherwise give up.

```

\expandafter\ifx\csname\f@family/\f@series/\f@shape\endcsname\relax
\family\default@family
\fi \fi

```

At this point a valid *family/series/shape* combination must have been found. We inform the user about this fact.

```

\font@warning{Using '\f@family/\f@series/\f@shape' instead}%

```

If we substitute a *family/series/shape* combination by the default, we don't want the warning to be printed out whenever this (unknown) combination is used. Therefore we globally \let the macro corresponding to the desired combination equal to its substitution. This requires the use of four \expandafter's since \csname... \endcsname has to be expanded before \@tempa (i.e. the requested combination), and this must happen before the \let is executed.

```
\global\expandafter\expandafter\expandafter\let\expandafter\@tempa
\csname\font@family/\font@series/\font@shape\endcsname
```

Now we can redefine \font@name accordingly.

```
\edef\font@name{\csname\font@family/\font@series/\font@shape/\font@size\endcsname}%
```

The last thing this macro does is to call \pickup@font again to load the font if it is not defined yet. At this point this code will loop endlessly if the defaults are not well defined.

```
\pickup@font}
```

**\strip@prefix** In \extract@font we will need a way to recover the replacement text of a macro. This is done by the primitive \meaning together with the macro \strip@prefix (for the details see appendix D of the T<sub>E</sub>Xbook, p. 382).

```
\def\strip@prefix#1>{}
```

**\extract@font** Here it comes: the macro solving all our problems (well, nearly all). What must this macro do? This is explained best with an example. Assume that *family* is 'cm', *series* is 'sansserif', *shape* 'normal', and *size* '12'. Assume further that this combination is defined, i.e. there exists the macro \cm/sansserif/normal. (Otherwise \extract@font doesn't get called.) Its replacement text consists of one (undefined) control sequence looking like

```
\<10>cmss10<12>cmss12<17>cmss17
```

For reasonable styles one usually needs more sizes but this is sufficient to get the flavour. We will define a macro \extract@fontinfo to find the external font name ('cmss12') for us:

```
\def\extract@fontinfo#1<12>#2<#3\@nil{%
\global\font\cm/sansserif/normal/12#2}
```

so that when it gets called via

```
\expandafter\extract@fontinfo
\string\<10>cmss10<12>cmss12<17>cmss17\@nil
```

#1 will contain all characters before <12>, #2 will be exactly cmss12, and #3 will be 17>cmss17. The expansion is therefore

```
\global\font\cm/sansserif/normal/12 cmss12
```

which is exactly what we want.

But this is only part of the whole story. It may be that the size requested does not occur in the \cm/sansserif/normal macro. And the simple definition of \extract@fontinfo we gave above does not allow us to specify the font substitution that we explained in 3.1.

Both problems are solved with the same trick: We define \extract@fontinfo as follows:

```
\def\extract@fontinfo#1<12>#2#3<#4\@nil{%
\global\font\cm/sansserif/normal/12
\ifcase 0#2#3\relax\or
#3 \font@warning{Size 12 not available
```

```

- using '#3' instead}\or
#3 \font@warning{Family/series/shape not available
- using '#3' instead}\else
\default@errfont \errmessage{Font not found}\fi}

```

How does this work? The first difference from the previous definition is that the characters of the external font name are split between parameters #2 and #3, #2 receiving only the first character. If this first character is not a digit, the `\ifcase` will get the 0 and select the first alternative. #2 and #3 are combined again and used as a file name. If #2 is a digit then the expansion of `\ifcase` will combine the 0 and #2 to a number.<sup>11</sup> Cases 1 and 2 select the second and third alternatives that use #3 as the substitution font.

The default case is reserved for a size that cannot be found in the tables. We achieve this by calling `\extract@fontinfo` via

```

\expandafter \extract@fontinfo
\string\<10>cmss10<12>cmss12<17>cmss17
<12>3\@nil

```

If the size ('12' in this case) appears in the `\<10>...` macro everything works as explained above, the only difference being that argument #4 of `\extract@fontinfo` additionally gets the `<12>3` tokens. However, if the size is not found, everything up to the final `<12>` is in argument #1, #2 gets 3, and #3 and #4 are empty. Therefore the `\ifcase` will select the default alternative and write an error message.

We have cheated a bit, of course. Normally digits and characters like `/<>` are not allowed as part of control sequences. Additionally the macros are hidden inside other control sequences so that we have to build `\extract@fontinfo` in several steps. Putting everything together we define `\extract@font` as follows.

```
\def\extract@font{%
```

`\@tempa` is made an abbreviation for the head of the definition of `\extract@fontinfo`.

```
\def\@tempa{\def\extract@fontinfo###1}%
```

Then we define `\@tempb` so that it expands to `<<size>>`. We use this slightly complicated construction to ensure that all characters have `\catcode 12`. This is needed for the delimiter matching in macro expansion.

```
\edef\@tempb{<\expandafter\strip@prefix\meaning\font@size>}%
```

Now we can define `\extract@fontinfo`.

```
\expandafter\@tempa\@tempb##2##3<##4\@nil{%
```

Remember that `\font@name` expands to the internal font name.

```
\global\expandafter\font \font@name
```

Here comes the `\ifcase`. For the benefit of the user, the warning messages are a bit more eloquent.

```

\ifcase0##2##3\relax\or
##3
\font@warning{Font/shape '\font@family/\font@series/\font@shape'
in size \@tempb\space not available}%
\font@warning{Using '##3' instead}\or
##3
\font@warning{Font/shape '\font@family/\font@series/\font@shape'
not available}%
\font@warning{Using '##3' instead}\else

```

<sup>11</sup> Recall that 01 is a valid *<number>* for  $\TeX$ .

There are two points to be explained here: `\default@errhelp` is the font to be selected if the requested size is not found in the tables. `\nofont@help` denotes a token register that contains a help message for the user. Its definition is given below.

```
\default@errfont \errhelp\nofont@help
\errmessage{Font \expandafter
\string\font@name\space
not found}%
\fi}%
```

Now we must extract the font information from the *family/series/shape* macro. This is done in two steps: first generate the macro name by `\csname...\endcsname` and expand it to get its replacement text. Then use `\string` to convert this text into a sequence of character tokens with `\catcode 12`. We define `\font@info` to contain this sequence followed by `<<size>>` (which is stored in `\@tempb`).

```
\edef\font@info{\expandafter\expandafter\expandafter\string
\csname \f@family/\f@series/\f@shape \endcsname \@tempb}%
```

Now we call `\extract@fontinfo`. Note the `3<\@nil` tokens at the end.

```
\expandafter\extract@fontinfo\font@info 3<\@nil
```

Finally we call the corresponding “extra” macro to finish things.

```
\csname extra//\f@family/\f@series \expandafter
\endcsname \font@name \relax}
```

The `\relax` at the end needs to be explained. This is inserted to prevent T<sub>E</sub>X from scanning too far when it is executing the replacement text of the “extra” macro.

`\nofont@help` `\nofont@help` is a token register containing a help message. It is defined using plain T<sub>E</sub>X’s `\newhelp` macro.

```
\newhelp\nofont@help
{You requested a font/series/shape/size combination that is
totally unknown. I have inserted a special font name
that will produce interesting effects in your output.
There are two cases in which this error can occur:
\space 1) You used the \string\size\space macro to select
a size that is not available.
\space 2) If you did not do that, go to your local ‘wizard’
and \spaces complain fiercely that the font
selection tables are corrupted!
(And do not worry about the missing escape characters in the
error traceback above!)
```

## 8 Assigning math fonts to *versions*

`\define@mathalphabet` We begin with the definition of the macro `\define@mathalphabet` which is built to append definitions specific to a new math *alphabet* to the replacement text of a *version* macro. It takes six arguments: the math *version* name (as a string of characters), a control sequence identifying the new math *alphabet*, the number of the new math *group* (normally a control sequence defined via `\countdef`), and finally three strings of characters denoting font *family*, *series*, and *shape*. If the shape parameter (`#6`) is empty then the *alphabet* `#2` is not available in *version* `#1`.

```
\def\define@mathalphabet#1#2#3#4#5#6{%
```

The first thing it does is to check if the name of the math *version* is already defined. This is the case if there already exist other math *alphabets* in this *version*. We must of course remember these definitions. To do so we save the contents of the macro in the token register `\toks@`.

```
\expandafter\ifx\csname #1\endcsname\relax
```

If there is no other *math alphabet* in this *version*, we simply store an empty token list in this register.

```
\toks@{}
```

Otherwise, we generate the control sequence denoting the macro using `\csname...` `\endcsname` and store its replacement text in `\toks@`. Note the three `\expandafter` primitives to achieve this.

```
\else
  \toks@\expandafter\expandafter\expandafter
    {\csname #1\endcsname}%
\fi
```

Depending on the shape parameter (#6) we have different things to do. We save the sequence of character tokens in a temporary control sequence.

```
\def\@tempa{#6}%
```

Now we globally redefine the *version*. Since the name of the *version* is given as a sequence of characters we must again build a macro name out of it. We use an `\xdef` so that the definition is expanded first.

```
\expandafter\xdef\csname#1\endcsname
```

This is necessary since we want to insert the contents of token register `\toks@`.

```
{\the\toks@
```

Then we append the new definitions for the *alphabet* #1. The `\noexpand` is necessary to insert the *math alphabet identifier* without expanding it.

```
\gdef\noexpand#2%
```

We must now catch the case that the shape parameter #6 saved in `\@tempa` is empty, i.e. that the *alphabet* is not available in this *version*. We simply include a call to the `\no@version@warning` defined earlier.

```
\ifx\@tempa\@empty
  {\noexpand\no@version@warning
   \noexpand\math@version
   \noexpand#2}%
```

Otherwise, we include a call to `\select@group` (see below) with the three arguments *math alphabet identifier*, *math group number*, and font *family/series/shape* definition macro.

```
\else
  {\noexpand\select@group
   \noexpand#2#3%
   \expandafter\noexpand\csname #4/#5/#6\endcsname}%
\fi}%
```

Now the macro switching to the *version* #1 contains a definition for *alphabet* #2. Finally we force a call to `\glb@settings` at the next time the fonts change by globally redefining `\glb@currsiz`.

```
\gdef\glb@currsiz{}
```

`\define@mathgroup` `\define@mathgroup` is similar to `\define@mathalphabet`. This macro is never called when processing a document, only during the font definition phase (e.g. by a style file in L<sup>A</sup>T<sub>E</sub>X or when dumping a format file). It is used for those *math groups* that are used via `\mathchardef` primitives. Since we don't need a *math alphabet identifier* to select those symbols, the macro takes only five arguments: the *math version* name as a sequence of character tokens, the *math group number* as a control sequence (it must already be allocated using `\new@mathgroup`) or as a digit (for *groups* 0 to 3, and font *family*, *series*, and *shape* name (as a sequence of character tokens). The first part is therefore completely analogous to the definition of `\define@mathalphabet`.

```
\def\define@mathgroup#1#2#3#4#5{%
```

```

\expandafter\ifx\csname#1\endcsname\relax
  \toks@{}%
\else
  \toks@\expandafter\expandafter\expandafter
  {\csname#1\endcsname}%
\fi

```

Since this code is never called by the user there is no need to issue a warning when the  $\langle math\ group\ number \rangle$  isn't allocated. However, the font tables must be defined consistently!<sup>12</sup> Instead of `\select@group` it uses `\getanddefine@fonts` which has only two arguments:  $\langle math\ group\ number \rangle$  and the font *family/series/shape* combination.

```

\expandafter\xdef\csname#1\endcsname
  {\the\toks@
  \noexpand\getanddefine@fonts#2%
  \expandafter\noexpand\csname #3/#4/#5\endcsname}%

```

The tail is literally the same as in `define@mathalphabet`.

```
\gdef\glb@currsize{}
```

`\getanddefine@fonts` `\getanddefine@fonts` has two arguments: the  $\langle math\ number\ number \rangle$  and the *family/series/shape* name as a control sequence.

```
\def\getanddefine@fonts#1#2{%
```

We append the current `\f@size` to #2 to obtain the font name.<sup>13</sup>

```
\edef\font@name{\csname \string#2/\f@size\endcsname}%
```

Then we call `\pickup@font` to load it if necessary. We remember the internal name as `\textfont@name`.

```
\pickup@font \let\textfont@name\font@name
```

Same game for `\scriptfont` and `\scriptscriptfont`:

```

\edef\font@name{\csname \string#2/\sf@size\endcsname}%
\pickup@font \let\scriptfont@name\font@name
\edef\font@name{\csname \string#2/\ssf@size\endcsname}%
\pickup@font

```

Then we append the new `\textfont...` assignments to the `\math@fonts`.

```

\edef\math@fonts{\math@fonts
  \textfont#1\textfont@name
  \scriptfont#1\scriptfont@name
  \scriptscriptfont#1\font@name}}

```

`\select@group` `\select@group` has three arguments: the new  $\langle math\ alphabet\ identifier \rangle$  (a control sequence), the  $\langle math\ group\ number \rangle$ , and the *family/series/shape* definition macro name. We first check if we are in math mode.

```
\def\select@group#1#2#3{\ifmmode
```

We do these things locally:

```
\bgroup
```

We set the math fonts for the *family* in question by calling `\getanddefine@fonts` in the correct environment.

```

\let\math@fonts\@empty \escapechar\m@ne
\getanddefine@fonts#2#3%

```

<sup>12</sup> Terrible harm will come to you if you don't do it right! A crowd of angry users might come to stone you!

<sup>13</sup> One might ask why this expansion does not generate a macro name that starts with an additional `\` character. The solution is that `\escapechar` is set to `-1` before `\getanddefine@fonts` is called.

We globally select the math fonts...

```
\globaldefs\@ne \math@fonts
```

... and close the group to restore \globaldefs and \escapechar.

```
\egroup
```

As long as no *size* or *version* change occurs, the *(math alphabet identifier)* should simply switch to the installed *group* instead of calling \select@group unnecessarily. So we globally redefine the first argument (the new *(math alphabet identifier)*) to expand into a \group switch and then select this *alphabet*. Note that this redefinition will be overwritten by the next call to a *version* macro.

```
\gdef#1{\group #2}#1%
```

If we are not in math mode nothing needs to be done.

```
\fi}
```

**\glb@settings** The macro \glb@settings globally selects all math fonts for the current size. The first thing it does is to open up a group.

```
\def\glb@settings{\begingroup
```

This is done to keep the following changes local: set the \escapechar to -1 and make \math@fonts to expand to nothing.

```
\escapechar \m@ne
\let\math@fonts\@empty
```

Why do we \let \math@fonts equal to \@empty at this point? When \glb@settings gains control, a size change was requested and all previous font assignments need to be replaced. Therefore the old values of the fonts are no longer needed. For every *group* the new assignments are appended to \math@fonts. Now we set the script size and scriptscript size.

```
\csname S@\f@size\endcsname
```

This also sets the define@mathfonts switch. If it is true, we must switch the math fonts. We execute the macro for the current math *version*. This sets \math@fonts to a list of \textfont... assignments.

```
\ifdefine@mathfonts \csname \math@version \endcsname \fi
```

Then we set \globaldefs to 1 so that all following changes are done globally.

```
\globaldefs\@ne
```

The math font assignments recorded in \math@fonts are executed, \glb@currsiz is set to the wanted \f@size, and the \baselineskip parameter is set accordingly by the macro \setnew@baselineskip and then multiplied by \baselinestretch.

```
\math@fonts
\let \glb@currsiz \f@size
\setnew@baselineskip
\baselineskip\baselinestretch\baselineskip
```

Then we set the \strutbox and \normalbaselineskip.

```
\setbox\strutbox\hbox{\vrule\@height.7\baselineskip
\@depth.3\baselineskip \width\z@}%
\normalbaselineskip\baselineskip
```

The macro ends by closing the group. This restores all parameters changed locally (including \globaldefs!) to their previous values.

```
\endgroup}
```

**\baselinestretch** In \glb@settings we used \baselinestretch as a factor when assigning a value to \baselineskip. We use 1 as a default (i.e. no stretch).

```
\def\baselinestretch{1}
```

## Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

<b>Symbols</b>		<b>E</b>	<code>\new@mathgroup</code> 224, <u>228</u>
<code>\@depth</code> . . . . .	227, 237	<code>\errhelp</code> . . . . .	234
<code>\@empty</code> . . . . .	227, 235-237	<code>\errmessage</code> . . . . .	234
<code>\@gobble</code> . . . . .	227, 228	<code>\escapechar</code> 231, 236, 237	
<code>\@height</code> . . . . .	227, 237	<code>\extra@def</code> . . . . .	224, <u>228</u>
<code>\@nomath</code> . . . . .	227	<code>\extract@font</code> . . . . .	<u>232</u>
<code>\@spaces</code> . . . . .	227, 234	<b>F</b>	
<code>\@width</code> . . . . .	227, 237	<code>\f@family</code> . . . . .	229
<b>A</b>		<code>\f@series</code> . . . . .	229
<code>\aftergroup</code> . . . . .	230	<code>\f@shape</code> . . . . .	229
<b>B</b>		<code>\f@size</code> . . . . .	229
<code>\baselineskip</code> . . . . .		<code>\fam</code> . . . . .	228
. . . . .	223, 229, 237	<code>\family</code> . . . . .	223, <u>229</u>
<code>\baselinestretch</code> . . . . .		<code>\font@warning</code> . . . . .	<u>227</u>
. . . . .	<u>237</u> , 237	<b>G</b>	
<b>D</b>		<code>\getanddefine@fonts</code>	
<code>\default@family</code> . . . . .	223	. . . . .	226, <u>236</u>
<code>\default@series</code> . . . . .	223	<code>\glb@currsizes</code> . . . . .	<u>230</u>
<code>\default@shape</code> . . . . .	223	<code>\glb@settings</code> . . . . .	<u>237</u>
<code>\define@mathalphabet</code>		<code>\globaldefs</code> . . . . .	237
. . . . .	224, 225, <u>234</u>	<code>\group</code> . . . . .	224, <u>228</u>
<code>\define@mathgroup</code> . . . . .		<b>I</b>	
. . . . .	224, 225, <u>235</u>	<code>\ifdefine@mathfonts</code> <u>229</u>	
<code>\define@mathsizes</code> . . . . .		<b>M</b>	
. . . . .	225, <u>229</u>	<code>\math@version</code> . . . . .	<u>230</u>
<code>\define@newfont</code> . . . . .	<u>231</u>	<code>\mathversion</code> . . . . .	223, <u>230</u>
<code>\define@nomathsize</code> . . . . .	225, <u>229</u>	<b>N</b>	
		<code>\new@fontshape</code> 223, <u>228</u>	
		<code>\new@mathgroup</code> 224, <u>228</u>	
		<code>\newfam</code> . . . . .	228
		<code>\newhelp</code> . . . . .	234
		<code>\newif</code> . . . . .	229
		<code>\no@version@warning</code>	
		. . . . .	226, <u>227</u>
		<code>\nofont@help</code> . . . . .	<u>234</u>
		<code>\normalbaselineskip</code> 237	
		<b>P</b>	
		<code>\pickup@font</code> . . . . .	<u>230</u>
		<code>\preload@sizes</code> 224, <u>228</u>	
		<b>S</b>	
		<code>\scriptfont</code> . . . . .	236
		<code>\scriptscriptfont</code> . . . . .	236
		<code>\select@group</code> . . . . .	226, <u>236</u>
		<code>\selectfont</code> . . . . .	223, <u>230</u>
		<code>\series</code> . . . . .	223, <u>229</u>
		<code>\setnew@baselineskip</code>	
		. . . . .	<u>229</u>
		<code>\shape</code> . . . . .	223, <u>229</u>
		<code>\size</code> . . . . .	223, <u>229</u>
		<code>\split@name</code> . . . . .	<u>230</u>
		<code>\strip@prefix</code> . . . . .	<u>232</u>
		<code>\strutbox</code> . . . . .	237
		<b>T</b>	
		<code>\textfont</code> . . . . .	236
		<b>W</b>	
		<code>\wrong@fontshape</code> . . . . .	<u>231</u>

◇ Frank Mittelbach  
 Rainer Schöpf  
 Fachbereich Mathematik  
 Universität Mainz  
 Staudinger Weg 9  
 D-6500 Mainz  
 Federal Republic of Germany  
 Bitnet: schoepf@dmznat51



# L<sup>A</sup>T<sub>E</sub>X

## A Bar Chart in L<sup>A</sup>T<sub>E</sub>X

Dezsó Nagy

Frequently, a bar chart type of presentation of results is very handy. However, if this requires outside assistance, then it may become a time-consuming operation. In the following, a simple scheme is presented, which allows one to make up such a chart in a very short time. The `picture` environment in L<sup>A</sup>T<sub>E</sub>X is used. The procedure was developed originally as a result of keeping track of expenses related to computing costs. Due to the simplicity of the program, modifications for other applications are easy to make.

The essence of the procedure is the `m` macro, which, for each month, writes out the text for identification, then draws the various representations of the data. The macro uses four parameters the last 3 of which are  $x$ -coordinates in a L<sup>A</sup>T<sub>E</sub>X `picture` environment. See the chart on the following page for an example. The parameters are:

- #1 This is the text for identification which could be the name of the month, or any number. This field is put to the left of the bars in a box, flush right. As can be seen from the example for July, font changing commands are possible.
- #2 This number is depicted by a solid bar #2 units wide 2 mm high.
- #3 This number is the coordinate of the right end of a line centered vertically on the solid bar above.
- #4 This number is the coordinate of the right end of an open box of height 2 mm.

All values of parameters 2-4 are horizontal displacements from the origin of the enclosing `picture` environment. Thus, a value of 25 for parameter #2 will produce a bar 25 mm long.

If a remark is needed at some position, this can be done easily: the offset in the example is 100 mm.

### Brief Description

Counters `yo` and `y1` are assigned and initialised to fix the positions of the starting boxes and horizontal line drawn by `m`. The unit length is also set to 1 mm. The macro `m` draws the required boxes then decreases the counters by 10 mm, i.e. gets ready for the next bar. It must be noted here that, because of this reset in the macro, if a remark is required for an entry, then it must be done *before* the bar is

drawn. The date, to be placed at the bottom of the diagram, can be done easily as shown.

The final part of the program is to display some explanation in a **Legend** box. Most of the steps are self evident. The placement of the **Legend** box depends on the data presented. The values -55 and -128 are obtained by trial and error.

Following is a complete listing of the input.

---

```

\documentstyle{article}

\setlength{\textwidth}{39pc}
\setlength{\textheight}{54pc}

\newcounter{dr}      \pagestyle{empty}
\newcounter{yo}      \setcounter{yo}{140}
\newcounter{yl}      \setcounter{yl}{141}
\setlength{\unitlength}{1mm}

\def\m#1#2#3#4{%
  \put(-5,{\arabic{yl}})}%
                                {\makebox(0,0)[r]{#1}}
  \put(0,{\arabic{yo}}){\framebox({#4},2){}}
  \put(0,{\arabic{yl}}){\line(1,0){#3}}
  \put({#3},{\arabic{yo}}){\line(0,1){2.1}}
  \linethickness{2.1mm}
  \put(0,{\arabic{yl}}){\line(1,0){#2}}
  \thinlines
  \addtocounter{yo}{-10}
  \addtocounter{yl}{-10}
}

\begin{document}

\begin{center}
\begin{picture}(130,180)
\put(0,165){\makebox(0,0)[l]{
  {\huge\bf Computing costs}}}
\put(0,150){\makebox(0,0){0}}
\multiput(10,150)(10,0){10}{\addtocounter
  {dr}{1000}\makebox(0,0){\arabic{dr}}}
\put(0,148){\line(1,0){100}}
\multiput(0,148)(10,0){11}{\line(0,-1){1}}
\m{Jan}{8.7}{10.8}{11.4}
\m{Feb}{23.1}{24.8}{27}
\put(100,{\arabic{yl}}){\makebox(0,0)[l]{
  {\em geoid}}}
\m{3}{72.8}{78.7}{81.4}
\m{April}{52.5}{60}{66.9}
\put(100,{\arabic{yl}}){\makebox(0,0)[l]{
  {\em Super '88}}}
\m{May}{20.3}{0}{0}
\put(100,{\arabic{yl}}){\makebox(0,0)[l]{

```

```

{\em Chapman Conf.}}
\m{Jun}{2.8}{2.8}{11.1}
\m{\em July}{19.5}{31.1}{31.1}
\addtocounter{yo}{-20}
\put(0,{\arabic{yo}}){\makebox(0,0)%
{\today}}

\begin{picture}(50,15)(-55,-128)
\put(3,7){\makebox(0,0){Legend :}}
\put(13,6){\framebox(30,2){}}
\put(13,7){\line(1,0){22}}
\put(35,6){\line(0,1){2}}
{\footnotesize
\put(13,11){\makebox(0,0)[l]{compute}}
\put(31,4){\makebox(0,0){disc-space}}
\put(43,11){\makebox(0,0)[r]{other}}}

\linethickness{2.1mm}
\put(13,7){\line(1,0){15}}
\linethickness{1pt}
\put(-5,0){\framebox(50,15){}}
\end{picture}

\end{picture}
\end{center}

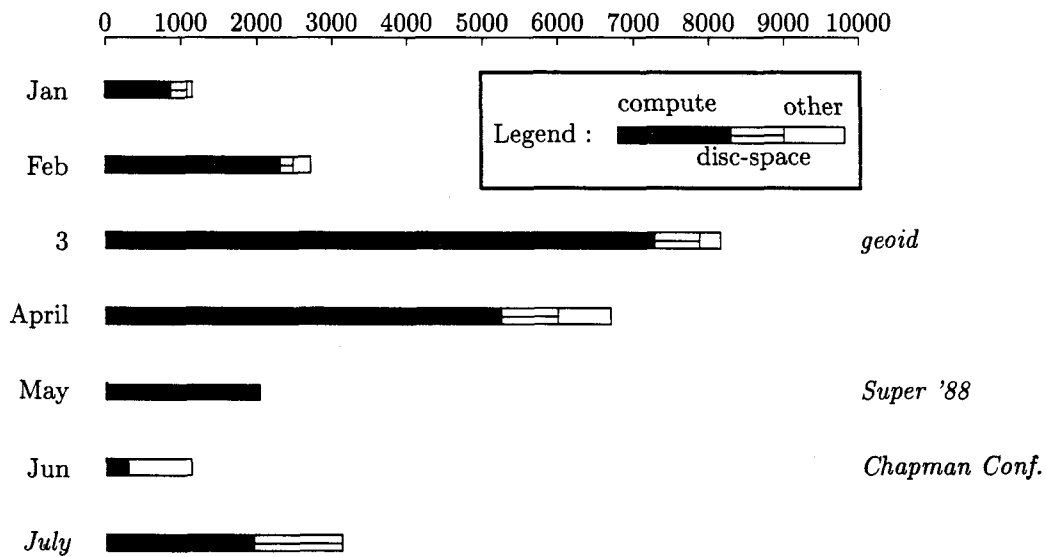
\vfill\ejct
\end{document}

```

---

◊ Dezső Nagy  
 Geological Survey of Canada  
 1 Observatory Crescent  
 Ottawa K1A 0Y3, Ontario  
 Canada  
 613-995-5449

## Computing costs



## Producing On-line Information Files with L<sup>A</sup>T<sub>E</sub>X

Hubert Partl  
Technische Universität Wien

### Printed Documents and On-line Information Files

Computer users nowadays expect that all necessary information about the computer is available in two forms: as printed documents, and as on-line information files.

- The printed documents are usually bought at the EDP center's book shop. They may be anything from short leaflets to complete books, and in any case they should be as beautiful and as readable as possible. Thus, T<sub>E</sub>X, together with one of its macro packages like L<sup>A</sup>T<sub>E</sub>X, and a laser printer are the ideal means to generate them.
- On-line information files are ASCII files stored on the computer itself. Whilst working on the computer, users can access them directly from the terminals on their own desks. Typically, the files are viewed on a terminal screen, or searched for certain keywords with an editor, or printed on a nearby cheap line printer.

For the authors and maintainers of the information texts, it is highly desirable that the same input file can be used to generate both the printed and the on-line versions. How can this be achieved in a L<sup>A</sup>T<sub>E</sub>X environment?

There are driver programs available that generate line printer or ASCII output from a DVI file, but these are aimed at proof-reading and previewing: They try hard to show how the text will eventually be broken into lines and pages in the final printed document. Their results as such are usually neither beautiful nor readable.

What we need is something different: We want the text to be set as beautiful and as readable as is possible in a simple line-oriented ASCII file, and with a layout that is best suited for the purpose of viewing on a terminal and printing on a line printer (e.g. 72 characters per line, 60 lines per page, blank lines to separate sections, and so on). We don't care about any relation to the line and page breaks of the printed version.

Obviously, we need two things to achieve this:

1. We need a L<sup>A</sup>T<sub>E</sub>X style or style option that will set our text such that the ASCII file will be as readable and as beautiful as possible. This will be dealt with in the present article.
2. We need an ASCII driver that will convert our DVI file into an ASCII file.

Several such drivers are available, most of them in the public domain. My favorite is Crudetype by R. M. Damerell (Royal Holloway and Bedford College, Egham, UK).

### My First Attempt

Here is my first attempt for such a document style option file, which I called `screen.sty`.

It is intended for generating ASCII file versions of non-mathematical texts, usually descriptions of computer programs and similar information.

Ideally, it should work like this:

- The printed manual is generated the usual way, with L<sup>A</sup>T<sub>E</sub>X and with a laser printer driver.
- The on-line version is generated by using a copy of the L<sup>A</sup>T<sub>E</sub>X input file, in which the option `screen` is added at the end of the option list in the `\documentstyle` command, and by feeding the resulting DVI file into an ASCII driver like Crudetype.

Real life, however, is a bit more complicated: I usually have to apply several manual changes to the copied version of the L<sup>A</sup>T<sub>E</sub>X input file and also to the generated output file. But even so, this is much easier than maintaining two completely different text files for the printed and on-line versions.

In spite of these drawbacks, I am presenting my humble first approach to the TUGBOAT readership—hoping that some readers can use my ideas, and that some will provide me with their ideas of how to do it better!

Now, let us have a look at the contents of my `screen.sty` file:

### Fonts

Only one "font" is available on a line printer, and it is a mono-spaced one. With T<sub>E</sub>X, this means that the whole document should be set in the typewriter font (`\tt`).

The normal-size typewriter font is selected by the following commands:

```
\normalsize\tt
```

This must be done at the beginning, so that `\baselineskip` and the `em` and `ex` units have the correct values for all length assignments to come.

All font changing commands are re-defined to refer to the `\tt` font:

```
\let\rm=\tt \let\bf=\tt \let\it=\tt
\let\sl=\tt \let\sf=\tt \let\sc=\tt
\let\em=\tt
```

Of course, this does not catch special fonts loaded by the user, nor implicit font changes other than `\em`, nor does it consider the mathematical

mode. If the L<sup>A</sup>T<sub>E</sub>X input file contains commands to load special fonts, these should be `\let` to `\tt`, too.

### Font Sizes

Only one character size is available on a line printer. All size changing commands are changed to refer to `\normalsize`, and instead of switching to the `\rm` font, they will switch to the `\tt` font.

First, we change the `\normalsize` command to switch to `\tt` rather than `\rm`:

```
\let\NormalsizeRm=\normalsize
\def\normalsize{\NormalsizeRm\tt}
```

Then, all other size changing commands are re-defined to this new `\normalsize` command:

```
\let\tiny=\normalsize
\let\scriptsize=\normalsize
\let\footnotesize=\normalsize
\let\small=\normalsize
\let\large=\normalsize
\let\Large=\normalsize
\let\LARGE=\normalsize
\let\huge=\normalsize
\let\Huge=\normalsize
```

### Accents and Special Characters

Overprinting should be avoided in the ASCII file generated, because it does not work when the file is viewed on a terminal screen. All accented and special characters have to be mapped to "normal" ASCII characters or character sequences.

Most accents can just be omitted, i.e. é can be printed as e, and so on.

```
\let\'=\relax \let\`=\relax \let\^=\relax
\let\c=\relax \let\~=\relax \let\==\relax
\let\.\=\relax \let\u=\relax \let\v=\relax
\let\H=\relax \let\d=\relax \let\b=\relax
\let\t=\relax
```

Various umlaut characters are to be replaced by two-character sequences, e.g. ä will be printed as ae, ß will be printed as ss, œ will be printed as oe, and so on. Most of these changes are straightforward:

```
\def\ss{ss} \def\aa{aa} \def\ae{ae}
\def\oe{oe} \def\AA{Aa} \def\AE{Ae}
\def\OE{Oe} \def\o{o} \def\O{Oe}
```

However, with the umlaut accent `\`, the following has to be considered: The german umlaut characters ä, ö, and ü are to be printed as ae, oe, and ue, but with the other letters, the umlaut dots can just be omitted, e.g. öe can be printed as oe, and ai can be printed as ai. Therefore, the `\` command is re-defined like this:

```
\def\"#1{\ifmmode #1}else
```

```
\if \string #1aa\else
\if \string #1oo\else
\if \string #1uu\else
\if \string #1AA\else
\if \string #1OO\else
\if \string #1UU\else
#1\fi \fi \fi \fi \fi \fi \fi}
```

Since we do not set any accents above letters, the dotless i and j can be replaced by their normal dotted versions:

```
\def\i{i} \def\j{j}
```

### Language Specific Modifications

Non-English speaking L<sup>A</sup>T<sub>E</sub>X users may use some modified versions of the L<sup>A</sup>T<sub>E</sub>X document styles. For instance, the `german` style option is likely to be used for german texts. In this case, some of the language specific definitions may need similar re-definitions.

If the option file `german.sty` has been processed, the active quotes character " must be re-defined with respect to the umlaut characters, the sharp s and the german quotes. The `\@ifundefined` command can be used to test whether the `german` option has been specified.

Here is an example how the french quotes ("guillemets") can be re-defined:

```
\def\flqq{{\tt <<}} \def\frqq{{\tt >>}}
```

### Mathematical Symbols

We do not consider the typesetting of mathematical formulae. However, even in non-mathematical texts, some symbols of T<sub>E</sub>X's math mode are used, e.g. dots, bullets, and arrows. These commands are re-defined to print appropriate ASCII characters or character sequences, switching to text mode and to the `\tt` font:

```
\def\ldots{\mbox{\tt ...}}
\let\cdots=\ldots
\let\dots=\ldots
\def\times{\mbox{\tt x}}
\def\bullet{\mbox{\tt *}}
\def\rightharpoonright{\mbox{\tt ->}}
\def\Rightarrow{\mbox{\tt =>}}
\def\longrightharpoonright{\mbox{\tt -->}}
\def\Longrightharpoonright{\mbox{\tt ==>}}
\def\leftarrow{\mbox{\tt <-}}
\def\Leftarrow{\mbox{\tt <=}}
\def\longleftarrow{\mbox{\tt <--}}
\def\Longleftarrow{\mbox{\tt <==}}
```

Other symbols should be added to this list, if needed.

### Other Special Characters

Two conversion problems still need to be solved:

In normal text, the character sequences -- and --- produce long dashes. Since we have switched to the \tt font, however, they produce two or three hyphens, respectively. It would be nice to find an automatic way that makes the character sequences -- and --- print a single hyphen (-) only.

A similar problem exists for the opening and closing quotes: In normal text, ‘ and ’ are used to print opening and closing quotes. Since we have switched to the \tt font, they produce double apostrophes. Instead, we would like them to print one double quotes character (").

I have not found a suitable definition yet that would accomplish this within T<sub>E</sub>X. I assume that it should involve \catcode to make the characters active, and \@ifnextchar to look at the following character. However, the hyphen sign and quotes should not become “fragile”, and the re-definitions should not inhibit the use of hyphens within the \hyphenation command nor the use of the grave character within the \catcode command...

Perhaps, a better way would be to define ligatures for verb—, ‘ and ’ in the TFM-File for font cmtt10. Of course, these ligatures should be disabled in the verbatim mode, i.e. they should be included in the \onoligs command.

### Kerning, Raising and Lowering

Both the character positions within each line and the line positions within each page are fixed in the line printer file. Therefore, kerning, raising, and lowering of characters must be avoided.

The following re-definitions make the \_ and ^ commands do nothing in mathematical mode:

```
\catcode'\_ =\active \let_ =\relax
\catcode'\^ =\active \let^ =\relax
```

The following re-definitions generate appropriate substitutes for the T<sub>E</sub>X and L<sup>A</sup>T<sub>E</sub>X logos:

```
\def\TeX{TeX} \def\LaTeX{LaTeX}
```

Similar re-definitions should be added for other logos of this kind, if needed.

### Vertical Skips

The files to be generated consist of discrete lines (as opposed to arbitrary character placement on the page). Therefore, all vertical skips must be integer multiples of the line height \baselineskip, and they must not be stretched or shrunk.

The \baselinestretch factor must be 1:

```
\def\baselinestretch{1}
```

If paragraphs are indented with no vertical skip, \parskip can be set to zero:

```
\parskip=0pt
```

If, however, they are not indented but is to be separated by a vertical skip, this skip should be one blank line:

```
\parskip=\baselineskip
```

The predefined vertical skips are re-defined to zero or one line, respectively:

```
\smallskipamount=0pt
\medskipamount=\baselineskip
\bigskipamount=\baselineskip
```

The sectioning commands are re-defined such that the vertical skips and their stretching are multiples of the line height, and that the heading is printed in \tt style:

```
\def\section{\@startsection
  {section}{1}{\z@}%
  {-2\baselineskip plus -2\baselineskip}%
  {1\baselineskip}%
  {\raggedright\normalsize\tt}}
\def\subsection{\@startsection
  {subsection}{2}{\z@}%
  {-1\baselineskip plus -1\baselineskip}%
  {1\baselineskip}%
  {\raggedright\normalsize\tt}}
\def\subsubsection{\@startsection
  {subsubsection}{3}{\z@}%
  {-1\baselineskip plus -1\baselineskip}%
  {1\baselineskip}%
  {\raggedright\normalsize\tt}}
```

Note that due to the \raggedbottom command (see below), the stretchable glue in these skips will *not* cause an actual stretching but will help L<sup>A</sup>T<sub>E</sub>X to find a suitable place for the page breaks.

The vertical skips used in all list environments are re-defined, too. Here is an example that re-defines them all to equal \parskip:

```
\def\@listI{\leftmargin\leftmarginI
  \topsep\z@ \parsep\parskip
  \itemsep\z@}
\let\@listi\@listI
\@listi
\def\@listii{\leftmargin\leftmarginii
  \labelwidth\leftmarginii
  \advance\labelwidth-\labelsep
  \topsep\z@ \parsep\parskip
  \itemsep\z@}
\def\@listiii{\leftmargin\leftmarginiii
  \labelwidth\leftmarginiii
  \advance\labelwidth-\labelsep
  \topsep\z@ \parsep\parskip
  \itemsep\z@}
```

Of course, these re-definitions don't catch explicit \vspace and \[\length] commands that ap-

pear in the L<sup>A</sup>T<sub>E</sub>X input file. These may have to be changed manually.

The `\raggedbottom` command makes sure that vertical skips are never stretched:

```
\raggedbottom
```

### Horizontal Skips

In an ASCII file, horizontal skips can only be accomplished by space characters which all have the fixed character width. Therefore, all horizontal skips must be multiples of the `\tt` font's character width. Note that 1 em is 2 character widths in this font.

If paragraphs are indented, `\parindent` should be set to something like:

```
\parindent=1em
```

If, however, they are not indented but are separated by a vertical skip only, `\parindent` is set to zero:

```
\parindent=0pt
```

The indentation amounts of all list environments are re-defined like this:

```
\leftmargini=2em
\leftmargin=\leftmargini
\leftmarginii=2em
\leftmarginiii=2em
\leftmarginiv=2em
```

The dot distance in the dotted lines within the table of contents has to be re-defined, too:

```
\def\@dottedtocline#1#2#3#4#5{%
  \ifnum #1>\c@tocdepth \else
  \vskip \z@ plus .2pt
  {\leftskip #2\relax \rightskip\@tocmarg
  \parfillskip -\rightskip
  \parindent #2\relax\@afterindenttrue
  \interlinepenalty\@M
  \leavevmode
  \@tempdima #3\relax
  \advance\leftskip \@tempdima
  \hbox{}\hskip -\leftskip
  #4\nobreak\leaders\hbox{\tt ~.~}\hfill
  \nobreak \hbox to\@pnumwidth
  {\hfil\rm #5}\par}\fi}
```

(This differs from the original definition only in the argument of the `\leaders` command.)

Of course, these re-definitions don't catch explicit `\hspace` or `\kern` commands that appear in the L<sup>A</sup>T<sub>E</sub>X input file. These may have to be changed manually.

The `\raggedright` command is needed, because in the `\tt` font, the spaces have fixed width and cannot be stretched for justification:

```
\raggedright
```

### Page Layout

The line width is set to 72 characters per line:

```
\textwidth=36em
```

The text height is set to 54 lines, which corresponds to 9 inches if the print density is 6 lines per inch:

```
\textheight=54\baselineskip
```

Other values (e.g. 80 characters per line) may be chosen in a similar way.

The top margin (including the header area) is set to zero:

```
\topmargin=0pt
\advance \topmargin by -\headheight
\advance \topmargin by -\headsep
```

The left margin, too, is set to zero for all pages:

```
\oddsidemargin=0pt
\evensidemargin=\oddsidemargin
```

The empty pagestyle is selected, because page numbers are normally not printed in on-line information files:

```
\pagestyle{empty}
```

However, for long documents, page numbers may be desirable. This can be accomplished by issuing an appropriate `\pagestyle` command in the L<sup>A</sup>T<sub>E</sub>X input file.

### Future Work

I am well aware that what I have presented here is a very first attempt only. Several extensions are certainly necessary to make it generally applicable. For instance, I have included the most simple text elements and environments only, but did not consider many other issues like footnotes, marginal notes, rules, tables, figures, title pages, abstracts, bibliographies, and so on. And I have completely omitted mathematics and pictures.

Furthermore, my re-definitions may contain bugs and errors, and perhaps a completely different approach would be much better.

It is my hope that many T<sub>E</sub>Xperts will now go on with this topic and propose and discuss different approaches — by e-mail, via the TeXhax mailing list, and in future editions of TUGBOAT. I am looking forward to many articles titled “*Another approach to producing on-line information files with L<sup>A</sup>T<sub>E</sub>X*”.

- ◊ Hubert Partl  
EDV-Zentrum  
Technische Universität Wien  
Wiedner Hauptstraße 8–10  
A-1040 Wien, Austria  
Bitnet: z3000pa@awituw01

**The doc-Option\***

Frank Mittelbach<sup>†</sup>  
Gutenberg Universität Mainz

**Abstract**

This style option contains the definitions that are necessary to format the documentation of style files. The style file was developed in Mainz in cooperation with the Royal Military College of Science.

**Contents**

<b>1 Introduction</b>	<b>245</b>	<b>3 The Description of Macros</b>	<b>251</b>
1.1 Using the doc style option	246	3.1 Macros surrounding the ‘definition parts’ . . . . .	251
<b>2 The User Interface</b>	<b>246</b>	3.2 Macros for the ‘documen- tation parts’ . . . . .	253
2.1 General conventions . . .	246	3.3 Formatting the margin . .	257
2.2 Describing the usage of new macros . . . . .	247	3.4 Creating index entries by scanning ‘macrocode’ . . .	257
2.3 Describing the definition of new macros . . . . .	247	3.5 Macros for scanning macro names . . . . .	259
2.4 Formatting the margins .	247	3.6 The index exclude list . .	262
2.5 Using a special escape character . . . . .	248	3.7 Macros for generating in- dex entries . . . . .	264
2.6 Cross-referencing all macros used . . . . .	248	3.8 Redefining the index envi- ronment . . . . .	266
2.7 Producing the actual in- dex entries . . . . .	248	3.9 Dealing with the change history . . . . .	268
2.8 Setting the index entries .	249	3.10 Bells and whistles . . . . .	269
2.9 Changing the default val- ues of style parameters . .	250	3.11 Layout parameters for doc- umenting style files . . . .	270
2.10 Additional bells and whis- tles . . . . .	250	3.12 Changing the \catcode of % . . . . .	271
2.11 Acknowledgements . . . .	251		

**1 Introduction**

The T<sub>E</sub>X macros which are described here allow definitions and documentation to be held in one and the same file. This has the advantage that normally very complicated instructions are made simpler to understand by comments inside the definition. In addition to this, updates are easier and only one source file needs to be changed. On the other hand, because of this, the style files are consid-

erably longer: thus T<sub>E</sub>X takes longer to load them. If this is a problem, there is an easy remedy: one needs only to write a small Pascal program that removes all lines that begin with a % sign.

The idea of integrated documentation was born with the development of the T<sub>E</sub>X program; it was crystallized in Pascal with the WEB system. The advantages of this method are plain to

\* This file has version number v1.5j dated 89/06/07. The documentation was last revised on 89/06/09.

<sup>†</sup> Further commentary added at Royal Military College of Science by B. HAMILTON KELLY; English translation of parts of the original German commentary provided by Andrew Mills.

see (it's easy to make comparisons [2]). Since this development, systems similar to WEB have been developed for other programming languages. But for one of the most complicated programming languages (TeX) the documentation has however been neglected. The TeX world seems to be divided between:—

- a couple of “wizards”, who produce many lines of completely unreadable code “off the cuff”, and
- many users who are amazed that it works just how they want it to do. Or rather, who despair that certain macros refuse to do what is expected of them.

I do not think that the WEB system is *the* reference work; on the contrary, it is a prototype which suffices for the development of programs within the TeX world. It is sufficient, but not totally sufficient.<sup>1</sup> As a result of WEB, new programming perspectives have been demonstrated; unfortunately, though, they haven't been developed further for other programming languages.

The method of documentation of TeX macros which I have introduced here

should also only be taken as a first sketch. It is designed explicitly to run under L<sup>A</sup>TeX alone. Not because I was of the opinion that this was the best starting point, but because from this starting point it was the quickest to develop.<sup>2</sup> As a result of this design decision, I had to move away from the concept of modularization; this was certainly a step backward.

I would be happy if this article could spark off discussion over TeX documentation. I can only advise anyone who thinks that they can cope without documentation to “Stop Time” until he or she completely understands the *AMS-TeX* source code.

### 1.1 Using the doc style option

Just like any other option, invoke it by including it amongst the style options in the optional parameter list for the `\documentstyle` command.

**N.B.** Because this style option makes the % character ignorable, which may interfere with the reading of other style options, doc should be the *last* style option invoked.

## 2 The User Interface

### 2.1 General conventions

A TeX file prepared to be used with the ‘doc’ style option consists of ‘documentation parts’ intermixed with ‘definition parts’.

Every line of a ‘documentation part’ starts with a percent sign (%) in column one. It may contain arbitrary TeX or L<sup>A</sup>TeX commands except that the character ‘%’ cannot be used as a comment character. To allow user comments, the `^^A` character is defined as a comment character later on.

All other parts of the file are called ‘definition parts’. They contain fractions of the macros described in the ‘documentation parts’.

If the file is used to define new macros (e.g. as a style file in the `\documentstyle` macro), the ‘documentation parts’ are bypassed at high speed and the macro definitions are pasted together, even if they are split into several ‘definition parts’.

**macrocode** On the other hand, if the documentation of these macros is to be produced, the ‘definition parts’ should be typeset verbatim. To achieve this, these parts are surrounded by the macrocode environment. More exactly: before a ‘definition part’ there should be a line containing

<sup>1</sup> I know that this will be seen differently by a few people, but this product should not be seen as the finished product, at least as far as applications concerning TeX are concerned. The long-standing debate over ‘multiple change files’ shows this well.

<sup>2</sup> This argument is a bad one; however, it is all too often trotted out.



```
%\begin{macrocode}
```

and after this part a line

```
%\end{macrocode}
```

There must be *exactly* four spaces between the % and `\end{macrocode}` — T<sub>E</sub>X is looking for this string and not for the macro while processing a ‘definition part’.

Inside a ‘definition part’ all T<sub>E</sub>X commands are allowed; even the percent sign could be used to suppress unwanted spaces etc.

`macrocode*` Instead of the `macrocode` environment one can also use the `macrocode*` environment which produces the same results except that spaces are printed as `□` characters.

## 2.2 Describing the usage of new macros

`\DescribeMacro` When you describe a new macro you may use `\DescribeMacro` to indicate that at this point the usage of a specific macro is explained. It takes one argument which will be printed in the margin and also produces a special index entry. For example, I used `\DescribeMacro{\DescribeMacro}` to make clear that this is the point where the usage of `\DescribeMacro` is explained.

`\DescribeEnv` An analogous macro `\DescribeEnv` should be used to indicate that a L<sup>A</sup>T<sub>E</sub>X environment is explained. It will produce a somewhat different index entry. Below I used `\DescribeEnv{verbatim}`.

`verbatim` It is often a good idea to include examples of the usage of new macros in the text. Because of the % sign in the first column of every row, the `verbatim` environment is slightly altered to suppress those characters.<sup>3</sup> The `verbatim*` environment is changed in the same way.

## 2.3 Describing the definition of new macros

`macro` To describe the definition of a new macro we use the `macro` environment. It has one argument: the name of the new macro.<sup>4</sup> This argument is also used to print the name in the margin and to produce an index entry. Actually the index entries for usage and definition are different to allow an easy reference. This environment might be nested. In this case the labels in the margin are placed under each other.

`\MacrocodeTopsep` There also exist four style parameters: `\MacrocodeTopsep` and `\MacroTopsep` are used to control the vertical spacing above and below the `macrocode` and the `macro` environment, `\MacroIndent` is used to indent the lines of code and `\MacroFont` holds the font and a possible size change command for the code lines. If you want to change their default values in a style file (like `ltugbot.sty`) use the `\DocstyleParms` command described below.

## 2.4 Formatting the margins

`\PrintDescribeMacro` As mentioned earlier, some macros and the `macro` environment print their arguments in the margin. This is actually done by three macros which are user definable.<sup>5</sup> They are named `\PrintDescribeMacro`, `\PrintDescribeEnv` and `\PrintMacroName` (called by the `macro` environment).

<sup>3</sup> These macros were written by Rainer Schöpf. He also provided a new `verbatim` environment which can be used inside of other macros. This will be documented elsewhere.

<sup>4</sup> This is a change to the style design I described in TUGboat 10#1 (Jan. 89). We finally decided that it would be better to use the macro name *with* the backslash as an argument.

<sup>5</sup> You may place the changed definitions in a separate style file or at the beginning of the documentation file. For example, if you don’t like any names in the margin but want a fine index you can simply `\let` these macros equal `\@gobble`. The `doc style` option won’t redefine any existing definition of these macros.

## 2.5 Using a special escape character

`\SpecialEscapechar` If one defines complicated macros it is sometimes necessary to introduce a new escape character because the ‘\’ has got a special `\catcode`. In this case one can use `\SpecialEscapechar` to indicate which character is actually used to play the rôle of the ‘\’. A scheme like this is needed because the macrocode environment and its counterpart `macrocode*` produce an index entry for every occurrence of a macro name. They would be very confused if you didn’t tell them that you’d changed `\catcodes`. The argument to `\SpecialEscapechar` is a single-letter control sequence, that is, one has to use `\|` for example to denote that ‘|’ is used as an escape character. `\SpecialEscapechar` only changes the behavior of the next macrocode or `macrocode*` environment.

The actual index entries created will all be printed with \ rather than |, but this probably reflects their usage, if not their definition, and anyway must be preferable to not having any entry at all. The entries *could* be formatted appropriately, but the effort is hardly worth it, and the resulting index might be more confusing (it would certainly be longer!).

## 2.6 Cross-referencing all macros used

`\DisableCrossrefs` As already mentioned, every new macro name used within a macrocode or `macrocode*`  
`\EnableCrossrefs` environment will produce an index entry. In this way one can easily find out where a specific macro is used. Since `TeX` is considerably slower when it has to produce such a bulk of index entries one can turn off this feature by using `\DisableCrossrefs` in the driver file. To turn it on again just use `\EnableCrossrefs`.<sup>6</sup>

`\DoNotIndex` But also finer control is provided. The `\DoNotIndex` macro takes a list of macro names separated by commas. Those names won’t show up in the index. You might use several `\DoNotIndex` commands: their lists will be concatenated. In this article I used `\DoNotIndex` for all<sup>7</sup> macros which are already defined in `LATeX`.

All three declarations are local to the current group.

## 2.7 Producing the actual index entries

Several of the aforementioned macros will produce some sort of index entries. These entries have to be sorted by an external program—the current implementation assumes that the `makeindex` program by Chen [4] is used.

But this isn’t built in: one has only to redefine some of the following macros to be able to use any other index program. Since the `doc style` option has to be the last option in the `\documentstyle` macro, all macros which are installation dependent are defined in such a way that they won’t overwrite a previous definition. Therefore it is safe to put the changed versions in a style file which might be read in before the `doc style` option.

To allow the user to change the specific characters recognized by his or her index program all characters which have special meaning in the `makeindex` program are given symbolic names.<sup>8</sup> However, all characters used should be of `\catcode` other than ‘letter’ (11).

`\actualchar` The `\actualchar` is used to separate the ‘key’ and the actual index entry. The  
`\quotechar` `\quotechar` is used before a special index program character to suppress its special  
`\encapchar` meaning. The `\encapchar` separates the indexing information from a letter string

<sup>6</sup> Actually, `\EnableCrossrefs` changes things more drastically; any following `\DisableCrossrefs` which might be present in the source will be ignored.

<sup>7</sup> In this implementation there is one exception: you can’t use `\par` in the argument of `\DoNotIndex`. This will be fixed in a later version.

<sup>8</sup> I don’t know if there exists a program which needs more command characters, but I hope not.

- which `makeindex` uses as a  $\TeX$  command to format the page number associated with a special entry. It is used in this style to apply the `\main` and the `\usage` commands.
- `\levelchar` Additionally `\levelchar` is used to separate 'item', 'subitem' and 'subsubitem' entries. It is a good idea to stick to these symbolic names even if you know which index program is used. In this way your files will be portable.
- `\SpecialMainIndex` To produce a main index entry for a macro the `\SpecialMainIndex` macro<sup>9</sup> may be used. It is called 'special' because it has to print its argument verbatim. If you want a normal index entry for a macro name `\SpecialIndex` might be used.<sup>10</sup> To index the usage of a macro or an environment `\SpecialUsageIndex` and `\SpecialEnvIndex` may be used. Additionally a `\SortIndex` command is provided. It takes two arguments—the sort key and the actual index entry.
- `\SpecialIndex`
- `\SpecialUsageIndex`
- `\SpecialEnvIndex`
- `\SortIndex`
- All these macros are normally used by other macros; you will need them only in an emergency.
- `\verbatimchar` But there is one characteristic worth mentioning: all macro names in the index are typeset with the `\verb*` command. Therefore one special character is needed to act as a delimiter for this command. To allow a change in this respect, again this character is referenced indirectly, by the macro `\verbatimchar`. It expands by default to `+` but if your code lines contain macros with '+' characters in their names (e.g. when you use `\+`) you will end up with an index entry containing `\verb+\++` which will be typeset as `\+` and not as `\+`. In this case you should redefine `\verbatimchar` globally or locally to overcome this problem.
- `\*` We also provide a `\*` macro. This is intended to be used for index entries like
- index entries  
Special macros for ~

Such an entry might be produced with the line:

```
\index{index entries\levelchar Special macros for \*}
```

## 2.8 Setting the index entries

- `theindex` Contrary to standard  $\LaTeX$ , the index is typeset in three columns by default. This is controlled by the  $\LaTeX$  counter 'IndexColumns' and can therefore be changed with a `\setcounter` declaration. Additionally one doesn't want to start a new page unnecessarily. Therefore the `theindex` environment is redefined. When the `theindex` environment starts it will measure how much space is left on the current page. If this is more than `\IndexMin` then the index will start on this page. Otherwise `\newpage` is called. Then a short introduction about the meaning of several index entries is typeset (still in onecolumn mode). Afterwards the actual index entries follow in multi-column mode.
- `\IndexMin`
- `\IndexPrologue` You can change this prologue with the help of the `\IndexPrologue` macro. Actually the section heading is also produced in this way, so you'd better write something like:
- ```
\IndexPrologue{\section*{Index} The index entries underlined ...}
```
- When the `theindex` environment is finished the last page will be reformatted to produce balanced columns. This improves the layout and allows the next article to start on the same page. Formatting of the index columns (values for `\columnsep` etc.) is controlled by the `\IndexParms` macro. It assigns the following values:
- ```
\parindent = 0.0pt           \columnsep = 15.0pt
\parskip    = 0.0pt plus 1.0pt \rightskip = 15.0pt
\mathsurround = 0.0pt        \parfillskip = -15.0pt
```
- `\@idxitem` Additionally it defines `\@idxitem` (which will be used when an `\item` command is

<sup>9</sup> This macro is called by the macro environment.

<sup>10</sup> This macro is called within the macrocode environment when encountering a macro name.

encountered) and selects `\small` size. If you want to change any of these values you have to define them all.

`\main` The page numbers for main index entries are encapsulated by the `\main` macro (underlining its argument) and the numbers denoting the description are encapsulated by the `\usage` macro (which produces *italics*). As usual these commands are user definable.

## 2.9 Changing the default values of style parameters

`\DocstyleParms` If you want to overwrite some default settings made by the doc style, you can either put your declarations in the driver file (that is after `doc.sty` is read in) or use a separate style file for doing this work. In the latter case you have to define the macro `\DocstyleParms` which should contain all assignments. This indirect approach is necessary because your style file will be read before the `doc.sty`, thus some of the registers are not then allocated. If you don't define this macro its default definition will be used which just starts the index process by calling `\makeindex`.

`\makeindex`

The doc style option currently assigns values to the following registers:

```

\IndexMin      = 80.0pt   \MacroTopsep    = 7.0pt plus 2.0pt minus 2.0pt
\marginparwidth = 96.0pt  \MacroIndent     = 10.0pt
\marginparpush  = 0.0pt   \MacrocodeTopsep = 3.0pt plus 1.2pt minus 1.0pt
\tolerance     = 1000

```

## 2.10 Additional bells and whistles

We provide macros for logos such as WEB,  $\mathcal{A}\mathcal{M}\mathcal{S}$ -TEX, BIBTEX, SLITEX and PLAIN TEX. Just type `\Web`, `\AmSTeX`, `\BibTeX`, `\SliTeX` or `\PlainTeX`, respectively.  $\LaTeX$  and  $\TeX$  are already defined in `latex.tex`.

`\meta` Another useful macro is `\meta` which has one argument and produces something like (*dimen parameter*).

`\OnlyDescription` You can use the `\OnlyDescription` declaration in the driver file to suppress the last part of your document. To make this work you have to place the command `\StopEventually` at a suitable point in your file. This macro has one argument in which you put all information you want to see printed if your document ends at this point (for example a bibliography which is normally printed at the very end). When the `\OnlyDescription` declaration is missing the `\StopEventually` macro saves its argument in a macro called `\Finale` which can afterwards be used to get things back (usually at the very end). Such a scheme makes changes in two places unnecessary.

`\StopEventually`

`\Finale`

Thus you can use this feature to produce a local guide for the  $\TeX$  users which describes only the usage of macros (most of them won't be interested in your definitions anyway). For the same reason the `\maketitle` command is slightly changed to allow multiple titles in one document. So you can make one driver file reading in several articles at once.

`\IndexListing` Last but not least I defined an `\IndexListing` macro which takes a file name as an argument and produces a verbatim listing of the file, indexing every command as it goes along. This might be handy, if you want to learn something about macros without enough documentation. I used this feature to cross-reference `latex.tex` getting a verbatim copy with about 15 pages index.<sup>11</sup>

`\changes` To maintain a change history within the file, the `\changes` command may be placed amongst the description part of the changed code. It takes three arguments, thus:

```

\changes{<version>}{<date>}{<text>}

```

<sup>11</sup> It took quite a long time and the resulting `.idx` file was longer than the `.dvi` file. Actually too long to be handled by the `makeindex` program directly (on our MicroVAX), but the final result was worth the trouble.

The changes may be used to produce an auxiliary file (L<sup>A</sup>T<sub>E</sub>X's `\glossary` mechanism is used for this) which may be printed after suitable formatting. The `\changes` macro encloses the *date* in parentheses and appends the *text* to form the printed entry in such a change history; because the `makeindex` program limits such fields to 64 characters, care should be taken not to exceed this limit when describing the change.

### 2.11 Acknowledgements

I would like to thank all folks at Mainz and at the Royal Military College of Science for their help in this project. Especially Brian and Rainer who pushed everything with their suggestions, bug fixes, etc.

## 3 The Description of Macros

As always, we begin by identifying the latest version of this file on the VDU and in the transcript file. But only if the macros are unknown to the system.

```
\@ifundefined{macro@cnt}{\endinput}
\typeout{Style-Option: 'doc' \fileversion\space <\filedate> (FMI)}
\typeout{English Documentation \spaces <\docdate> (RMCS and FMI)}
```

This time we also add a warning for the user.

```
\typeout{\spaces Warning: This style option should be used as last option}
\typeout{\spaces Warning: in the \protect\documentstyle\space command !}
```

`\fileversion` As you can see I used macros like `\fileversion` to denote the version number and the  
`\filedate` date. They are defined at the very beginning of the style file (without a surrounding  
`\docdate` macrocode environment), so I don't have to search for this place here when I change the version number. You can see their actual outcome in a footnote to the title.

The first thing that we do next is to get ourselves a new comment sign. Because all sensible signs are already occupied, we will choose one that can only be entered indirectly:

```
\catcode'\^^A=14
```

### 3.1 Macros surrounding the 'definition parts'

`\macrocode` Parts of the macro definition will be surrounded by the environment macrocode. Put more precisely, they will be enclosed by a macro whose argument (the text to be set 'verbatim') is terminated by the string `%\end{macrocode}`. Carefully note the number of spaces. `\macrocode` is defined completely analogously to `\verbatim`, but because a few small changes were carried out, almost all internal macros have got new names. We start by calling the macro `\macro@code`, the macro which bears the brunt of most of the work, such as `\catcode` reassignments, etc.

```
\def\macrocode{\macro@code
```

Then we take care that all spaces have the same width, and that they are not discarded.

```
\frenchspacing \@vobeyspaces
```

Before closing, we need to call `\xmacro@code`. It is this macro that expects an argument which is terminated by the above string. This way it is possible to keep the `\catcode` changes local.

```
\xmacro@code}
```

`\macro@code` We will now begin with the macro that does the actual work:

```
\def\macro@code{%
```

In theory it should consist of a `trivlist` environment, but the empty space before and after the environment should not be too large.

```
\topsep \MacrocodeTopsep
```

The next parameter we set is `\@beginparpenalty`, in order to prevent a page break before such an environment.

```
\@beginparpenalty \predisdisplaypenalty
```

We then start a `\trivlist`, set `\parskip` back to zero and start an empty `\item`.

```
\trivlist \parskip \z@ \item[]%
```

Additionally, everything should be set in `typewriter` font. Some people might prefer it somewhat differently; because of this the font choice is macro-driven.<sup>12</sup>

```
\MacroFont
```

Because `\item` sets various parameters, we have found it necessary to alter some of these retrospectively.

```
\leftskip\@totalleftmargin \advance\leftskip\MacroIndent
\rightskip\z@ \parindent\z@ \parfillskip\@flushglue
```

The next line consists of the L<sup>A</sup>T<sub>E</sub>X definition of `\par` used in `\verbatim` and should result in blank lines being shown as blank lines.

```
\blank@linefalse \def\par{\ifblank@line\hbox{}}\fi\blank@linetrue\@par}
```

What use is this definition of `\par`? We use the macro `\obeylines` of [3] which changes `^~M` to `\par` so that each line can control its own indentation. Next we must also ensure that all special signs are normalized; that is, they must be given `\catcode 12`.

```
\obeylines \let\do\@makeother \catcode'\active \@noligs \dospecials
```

We also initialize the cross-referencing feature by calling `\init@crossref`. This will start the scanning mechanism when encountering an escape character.

```
\init@crossref}
```

```
\ifblank@line \ifblank@line is the switch used in the definition above. In the original verbatim
\blank@linetrue environment the \if@tempswa switch is used. This is dangerous because its value may
\blank@linefalse change while processing lines in the macrocode environment.
```

```
\newif\ifblank@line
```

```
\endmacrocode Because we have begun a trivlist environment in the macrocode environment, we must
also end it. This is easily done using the following line of code:
```

```
\def\endmacrocode{\endtrivlist
```

Additionally `\close@crossref` is used to do anything needed to end the cross-referencing mechanism.

```
\close@crossref}
```

```
\MacrocodeTopsep In the code above, we have used two registers. Therefore we have to allocate them.
\MacroIndent The default values might be overwritten with the help of the \DocstyleParms macro.
```

```
\newskip\MacrocodeTopsep \MacrocodeTopsep = 3pt plus 1.2pt minus 1pt
\newdimen\MacroIndent \MacroIndent = 10pt
```

```
\MacroFont Here is the default definition for this macro:
```

```
\@ifundefined{MacroFont}{\def\MacroFont{\small\tt}}{}
```

<sup>12</sup>The font change has to be placed *after* the `\item`. Otherwise a change to `\baselineskip` will affect the paragraph above.

`\macrocode*` Just as with the verbatim environment, there is also a ‘star’ variant of the macrocode environment in which a space is shown by the symbol `_`. Until this moment, I have not yet used it (it will be used in the description of the definition of `\xmacro@code` below) but it’s exactly on this one occasion *here* that you can’t use it (cf. Münchhausen’s Marsh problem)<sup>13</sup> directly. Because of this, on this one occasion we’ll cheat around the problem with an additional comment character. But now back to `\macrocode*`. We start with the macro `\macro@code` which prepares everything and then call the macro `\sxmacro@code` whose argument is terminated by the string `%_UUUU\end{macrocode*}`.

```
\@namedef{macrocode*}{\macro@code\sxmacro@code}
```

As we know, `\sxmacro@code` and then `\end{macrocode*}` (the macro, not the string), will be executed, so that for a happy ending we still need to define the macro `\endmacrocode*`.

```
\expandafter\let\csname endmacrocode*\endcsname = \endmacrocode
```

`\xmacro@code` As already mentioned, the macro `\xmacro@code` expects an argument delimited by the string `%_UUUU\end{macrocode}`. At the moment that this macro is called, the `\catcode` of TeX’s special characters are 12 (‘other’) or 13 (‘active’). Because of this we need to utilize a different escape character during the definition. This happens locally.

```
\begingroup
\catcode'\|=z@_ \catcode'\[=\@ne_ \catcode'\]=\tw@
```

Additionally, we need to ensure that the symbols in the above string contain the `\catcodes` which are available within the macrocode environment.

```
\catcode'\{=12_ \catcode'\}=12
\catcode'\%=12_ \catcode'\_=\active_ \catcode'\]=\active
```

Next follows the actual definition of `\macro@code`; notice the use of the new escape character. We manage to get the argument surrounded by the string `\end{macrocode}`, but at the end however, in spite of the actual characters used during the definition of this macro, `\end` with the argument `{macrocode}` will be executed, to ensure a balanced environment.

```
|gdef|xmacro@code#1%_UUUU\end{macrocode}[#1\end{macrocode}]
```

`\sxmacro@code` The definition of `\sxmacro@code` is completely analogous, only here a slightly different terminating string will be used. Note that the space is not active in this environment.

```
|catcode'| =12
|gdef|sxmacro@code#1%   \end{macrocode*}[#1\end{macrocode*}]
```

Because the `\catcode` changes have been made local by commencing a new group, there now follows the matching `\endgroup` in a rather unusual style of writing.

```
|endgroup
```

### 3.2 Macros for the ‘documentation parts’

`\DescribeMacro` `\DescribeEnv` The `\DescribeMacro` and `\DescribeEnv` macros should print their arguments in the margin and produce an index entry. We simply use `\marginpar` to get the desired result. This is however not the best solution because the labels might be slightly misplaced. One also might get a lot of ‘marginpar moved’ messages which are hard-wired into the L<sup>A</sup>T<sub>E</sub>X output routine.<sup>14</sup> First we change to horizontal mode if necessary.

<sup>13</sup> Karl Friedrich Hieronymus Frhr. v. Münchhausen (\*1720, †1797). Several books were written about fantastic adventures supposedly told by him (see [5] or [1]). In one story he escaped from the marsh by pulling himself out by his hair.

<sup>14</sup> It might be better to change these macros into environments like the macro environment.

The L<sup>A</sup>T<sub>E</sub>X macros `\@bsphack` and `\@esphack` are used to make those commands invisible (i.e. to normalize the surrounding space and to make the `\spacefactor` transparent).

```
\def\DescribeMacro#1{\leavevmode\@bsphack
\marginpar{\raggedleft\PrintDescribeMacro{#1}}%
```

Note the use of `\raggedleft` to place the output flushed right. Finally we call a macro which produces the actual index entry and finish with `\@esphack` to leave no trace.<sup>15</sup>

```
\SpecialUsageIndex{#1}\@esphack\ignorespaces}
```

The `\DescribeEnv` macro is completely analogous.

```
\def\DescribeEnv#1{\leavevmode\@bsphack
\marginpar{\raggedleft\PrintDescribeEnv{#1}}%
\SpecialEnvIndex{#1}\@esphack\ignorespaces}
```

To put the labels in the left margin we have to use the `\reversemarginpar` declaration. (This means that the `doc.sty` can't be used with all style options.) We also make the `\marginparpush` zero and `\marginparwidth` suitably wide.

```
\reversemarginpar
\setlength\marginparpush{0pt} \setlength\marginparwidth{8pc}
```

`\bslash` From time to time, it is necessary to print a `\` without being able to use the `\verb` command because the `\catcodes` of the symbols are already firmly established. In this instance we can use the command `\bslash` presupposing, of course, that the actual font in use at this point contains a 'backslash' as a symbol. Note that this definition of `\bslash` is expandable; it inserts a `\_12`. This means that you have to `\protect` it if it is used in 'moving arguments'.

We start a new group in which to hide the alteration of `\catcodes`, and make `|` introduce commands, whilst `\` becomes an 'other' character.

```
{\catcode'\|=z@ \catcode'\|=12
```

Now we are able to define `\bslash` (globally) to generate a backslash of `\catcode` 'other'. We then close this group, restoring original `\catcodes`.

```
|gdef\bslash{\}}
```

`\verbatim` The verbatim environment holds no secrets; it consists of the normal L<sup>A</sup>T<sub>E</sub>X environment. We also set the `\@beginparpenalty` and change to the font given by `\MacroFont`.

```
\def\verbatim{\@beginparpenalty \predisplaypenalty \@verbatim
\MacroFont \frenchspacing \vobeyspaces \@xverbatim}
```

`\@verbatim` Additionally we redefine the `\@verbatim` macro so that it suppresses `%` characters at the beginning of the line. The first lines are copied literally from `latex.tex`.

```
\def\@verbatim{\trivlist \item[]\if@minipage\else\vskip\parskip\fi
\leftskip\@totalleftmargin\rightskipz@
\parindentz@\parfillskip\@flushglue\parskipz@
\@tempwafalse
```

`\@verbatim` sets `^^M`, the end of line character, to be equal to `\par`. This control sequence is redefined here; `\@@par` is the paragraph primitive of T<sub>E</sub>X.

```
\def\par{\if@tempwa\hbox{}}\fi\@tempwattrue\@@par
```

<sup>15</sup> The whole mechanism won't work because of the `\leavevmode` in front. As a temporary change `\ignorespaces` is added.



We add to the definition of `\par` a control sequence, `\check@percent`, whose task it is to check for a percent character.

```
\check@percent}%
```

The rest is again copied literally from `latex.tex`.

```
\obeylines \tt \catcode'\active \noligs \let\do\makeother \dospecials}
```

`\check@percent` Finally we define `\check@percent`. Since this must compare a character with a percent sign we must first (locally) change percent's `\catcode` so that it is seen by T<sub>E</sub>X. The definition itself is nearly trivial: grab the following character, check if it is a %, and insert it again if not. At the end of the `verbatim` environment this macro will peek at the next input line. In the case the argument to `\check@percent` might be a `\par` or a macro with arguments. Therefore we make the definition `\long` (`\par` allowed) and use the normal `\next` mechanism to reinsert the argument after the `\fi` if necessary.

```
{\catcode'\%=12
\long\gdef\check@percent#1{\ifx #1%\let\next\empty \else
\let\next#1\fi \next}}
```

`\macro` The macro environment is implemented as a trivlist environment, whereby in order  
`\macro@` that the macro names can be placed under one another in the margin (corresponding  
`\macro@cnt` to the macro's nesting depth), the macro `\makelabel` must be altered. In order to store the nesting depth, we use a counter.

```
\newcount\macro@cnt \macro@cnt=0
```

The environment takes an argument—the macro name to be described. Since this name may contain special 'letters' we have to re-`\catcode` them before scanning the argument. This is done by the `\MakePrivateLetters` macro.

```
\def\macro@\begingroup \MakePrivateLetters \macro@}
```

After scanning the argument we close the group to get the normal `\catcodes` back. Then we assign a special value to `\topsep` and start a trivlist environment.

```
\long\def\macro@#1{\endgroup \topsep\MacroTopsep \trivlist
```

We also save the name being described in `\saved@macroname` for use in conjunction with the `\changes` macro.

```
\edef\saved@macroname{\string#1}%
```

Now there follows a variation of `\makelabel` which is used should the environment not be nested, or should it lie between two successive `\begin{macro}` instructions or explanatory text. One can recognize this with the switch `\if@inlabel` which will be true in the case of successive `\item` commands.

```
\def\makelabel##1{\llap{##1}}%
```

If `@inlabel` is true and if `\macro@cnt > 0` then the above definition needs to be changed, because in this case I<sup>A</sup>T<sub>E</sub>X would otherwise put the labels all on the same line and this would lead to them being overprinted on top of each other. Because of this `\makelabel` needs to be redefined in this case.

```
\if@inlabel
```

If `\macro@cnt` has the value 1, then we redefine `\makelabel` so that the label will be positioned in the second line of the margin. As a result of this, two macro names appear correctly, one under the other. It's important whilst doing this that the generated label box is not allowed to have more depth than a normal line since otherwise the distance between the first two text lines of T<sub>E</sub>X will be incorrectly calculated. The definition should then look like:

```
\def\makelabel##1{\llap{\vtop to \baselineskip
{\hbox{\strut}\hbox{##1}\vss}}}
```

Completely analogous to this is the case where labels need to be placed one under the other. The lines above are only an example typeset with the `verbatim` environment. To produce the real definition we save the value of `\macro@cnt` in `\count@` and empty the temp macro `\@tempa` for later use.

```
\let\@tempa\@empty \count@\macro@cnt
```

In the following loop we append for every already typeset label an `\hbox{\strut}` to the definition of `\@tempa`.

```
\loop \ifnum\count@>\z@
  \edef\@tempa{\@tempa\hbox{\strut}}\advance\count@\m@ne \repeat
```

Now we put the definition of `\makelabel` together.

```
\edef\makelabel##1{\llap{\vtop to\baselineskip
  {\@tempa\hbox{##1}\vss}}}%
```

Next we increment the value of the nesting depth counter. This value inside the macro environment is always at least one after this point, but its toplevel definition is zero. Provided this environment has been used correctly, `\macro@cnt = 0` should not occur when `@inlabel = true`. It is however possible if this environment is used within other list environments (but this would have little point).

```
\advance \macro@cnt \@ne
```

If `@inlabel` is false we reset `\macro@cnt` assuming that there is enough room to print the macro name without shifting.

```
\else \macro@cnt\@ne \fi
```

Now the label will be produced using `\item`. The following line is only a hack saving the day until a better solution is implemented. We have to face two problems: the argument might be a `\par` which is forbidden in the argument of other macros if they are not defined as `\long`, or it is something like `\iffalse` or `\else`, i.e. something which will be misinterpreted when `TEX` is skipping conditional text. In both cases `\item` will bomb, so we protect the argument by using `\string`.

```
\edef\@tempa{\noexpand\item[\noexpand\PrintMacroName{\string#1}]\@tempa
```

At this point we also produce an index entry. Because it is not known which index sorting program will be used, we do not use the command `\index`, but rather a command `\SpecialMainIndex`. This may be redefined by the user in order to generate an index entry which will be understood by the index program in use (note the definition of `\SpecialMainIndex` for our installation).

```
\SpecialMainIndex{#1}\nobreak
```

The `\nobreak` is needed to prevent a page break after the `\write` produced by the `\SpecialMainIndex` macro. We exclude the new macro in the cross-referencing feature, to prevent spurious non-main entry references. Again we have to watch out for problematic arguments. In case of `\par` we wait for a new implementation; the conditionals are uncritical.

```
\def\@tempa{#1}%
\ifx\@tempa\defpar \else \DoNotIndex{#1}\fi
```

Because the space symbol should be ignored between the `\begin{macro}{...}` and the following text we must take care of this with `\ignorespaces`.

```
\ignorespaces}
```

`\endmacro` At this command nothing special needs to happen. However, the `trivlist` environment must still be ended. Because of the `\endgroup` which is used by `\end`, the changes to `\macro@cnt` stay local to the environment.

```
\let\endmacro\endtrivlist
```

`\MacroTopsep` Here is the default value for the `\MacroTopsep` parameter used above.

```
\newskip\MacroTopsep      \MacroTopsep = 7pt plus 2pt minus 2pt
```

### 3.3 Formatting the margin

The following three macros should be user definable. Therefore we define those macros only if they have not already been defined.

`\PrintMacroName` The formatting of the macro name in the left margin is done by these macros. We first set a `\string` to get the height and depth of the normal lines. Then we change to the `\MacroFont` using `\string` to `\catcode` the argument to other (assuming that it is a macro name). Finally we print a space. The font change remains local since this macro will be called inside an `\hbox`.

```
\@ifundefined{PrintMacroName}
  {\def\PrintMacroName#1{\string \MacroFont \string #1\ }}{}
```

We use the same formatting conventions when describing a macro.

```
\@ifundefined{PrintDescribeMacro}
  {\def\PrintDescribeMacro#1{\string \MacroFont \string #1\ }}{}
```

To format the name of a new environment there is no need to use `\string`.

```
\@ifundefined{PrintDescribeEnv}
  {\def\PrintDescribeEnv#1{\string \MacroFont #1\ }}{}
```

### 3.4 Creating index entries by scanning ‘macrocode’

When this doc option is used, it automatically invokes `\makeindex` to cause an `.idx` file to be generated. The following macros ensure that index entries are created for each occurrence of a TeX-like command (something starting with `\`). With the default definitions of `\SpecialMainIndex`, etc., the index file generated is intended to be processed by Chen’s `makeindex` program [4].

Of course, in *this* style file itself we’ve sometimes had to make `|` take the rôle of TeX’s escape character to introduce command names at places where `\` has to belong to some other category. Therefore, we may also need to recognize `|` as the introducer for a command when setting the text inside the macrocode environment. Other users may have the need to make similar reassignments for their macros.

`\SpecialEscapechar` The macro `\SpecialEscapechar` is used to denote a special escape character for the next macrocode environment. It has one argument—the new escape character given as a ‘single-letter’ control sequence. Its main purpose is defining `\special@escape@char` to produce the chosen escape character `\catcode` d to 12 and `\active@escape@char` to produce the same character but with `\catcode` 13.

The macro `\special@escape@char` is used to *print* the escape character while `\active@escape@char` is needed in the definition of `\init@crossref` to start the scanning mechanism.

In the definition of `\SpecialEscapechar` we need an arbitrary character with `\catcode` 13. We use `‘` and ensure that it is active. The `\begingroup` is used to make a possible change local to the expansion of `\SpecialEscapechar`.

```
\def\SpecialEscapechar#1{%
  \begingroup \catcode‘\~\active
```

Now we are ready for the definition of `\active@escape@char`. It’s a little tricky: we first define locally the uppercase code of `‘` to be the new escape character.

```
\uccode‘\~#1%
```

Around the definition of `\active@escape@char` we place an `\uppercase` command. Recall that the expansion of `\uppercase` changes characters according to their `\uccode`, but leaves their `\catcodes` untouched (cf. `TEXbook` page 41).

```
\uppercase{\gdef\active@escape@char{~}}%
```

The definition of `\special@escape@char` is easier; we use `\string` to `\catcode` the argument of `\SpecialEscapechar` to 12 and suppress the preceding `\escapechar`.

```
\escapechar\mOne \xdef\special@escape@char{\string#1}%
```

Now we close the group and end the definition: the value of `\escapechar` as well as the `\uccode` and `\catcode` of `'~'` will be restored.

```
\endgroup}
```

`\init@crossref` The replacement text of `\init@crossref` should fulfil the following tasks:

- 1) `\catcode` all characters used in macro names to 11 (i.e. 'letter').
- 2) `\catcode` the `'\'` character to 13 (i.e. 'active').
- 3a) `\let` the `'\'` equal `\scan@macro` (i.e. start the macro scanning mechanism) if there is no special escape character (i.e. the `\special@escape@char` is `'\'`).
- 3b) Otherwise `\let` it equal `\bslash`, i.e. produce a printable `\`.
- 4) Make the *(special escape character)* active.
- 5) `\let` the active version of the special escape character (i.e. the expansion of `\active@escape@char`) equal `\scan@macro`.

The reader might ask why we bother to `\catcode` the `'\'` first to 12 (at the end of `\macro@code`) then re-`\catcode` it to 13 in order to produce a `\12` in case 3b) above. This is done because we have to ensure that `'\'` has `\catcode` 13 within the macrocode environment. Otherwise the delimiter for the argument of `\xmacro@code` would not be found (parameter matching depends on `\catcodes`).

Therefore we first re-`\catcode` some characters.

```
\begingroup \catcode'\|=z@ \catcode'\|=active
```

We carry out tasks 2) and 3b) first.

```
|gdef|init@crossref{|catcode'\|=active |let|\bslash
```

Because of the popularity of the `'@'` character as a 'letter' in macros, we normally have to change its `\catcode` here, and thus fulfil task 1). But the macro designer might use other characters as private letters as well, so we use a macro to do the `\catcode` switching.

```
|MakePrivateLetters
```

Now we `\catcode` the special escape character to 13 and `\let` it equal `\scan@macro`, i.e. fulfil tasks 4) and 5). Note the use of `\expandafter` to insert the chosen escape character saved in `\special@escape@char` and `\active@escape@char`.

```
|catcode|expandafter'|special@escape@char|active
|expandafter|let|active@escape@char|scan@macro}
|endgroup
```

If there is no special escape character, i.e. if `\SpecialEscapechar` is `\\`, the second last line will overwrite the previous definition of `\13`. In this way all tasks are fulfilled. For happy documenting we give default values to `\special@escape@char` and `\active@escape@char` with the following line:

```
\SpecialEscapechar{\\}
```

**\MakePrivateLetters** Here is the default definition of this command, which makes just the @ into a letter. The user may change it if he/she needs more or other characters masquerading as letters.

```
\ifundefined{MakePrivateLetters}
{\let\MakePrivateLetters\makeatletter}{}
```

**\close@crossref** At the end of a cross-referencing part we prepare ourselves for the next one by setting the escape character to \.

```
\def\close@crossref{\SpecialEscapechar\}
```

### 3.5 Macros for scanning macro names

**\scan@macro** The **\init@crossref** will have made **\active** our **\special@escape@char**, so that  
**\macro@namepart** each **\active@escape@char** will invoke **\scan@macro** when within the macrocode environment. By this means, we can automatically add index entries for every TeX-like command which is met whilst setting (in verbatim) the contents of macrocode environments.

```
\def\scan@macro{%
```

First we output the character which triggered this macro. Its version **\catcode** d to 12 is saved in **\special@escape@char**.

```
\special@escape@char
```

If the macrocode environment contains, for example, the command **\**, the second **\** should not start the scanning mechanism. Therefore we use a switch to decide if scanning of macro names is allowed.

```
\ifscan@allowed
```

The macro assembles the letters forming a TeX command in **\macro@namepart** so this is initially cleared; we then set **\next** to the *first* character following the **\** and call **\macro@switch** to determine whether that character is a letter or not.

```
\let\macro@namepart\@empty
\def\next{\futurelet\next\macro@switch}%
```

As you recognize, we actually did something else, because we have to defer the **\futurelet** call until after the final **\fi**. If, on the other hand, the scanning is disabled we simply **\let \next** equal 'empty'.

```
\else \let\next\@empty \fi
```

Now we invoke **\next** to carry out what's needed.

```
\next}
```

**\ifscan@allowed** **\ifscan@allowed** is the switch used above to determine if the **\active@escape@char**  
**\scan@allowedtrue** should start the macro scanning mechanism.  
**\scan@allowedfalse** **\newif\ifscan@allowed \scan@allowedtrue**

**\EnableCrossrefs** At this point we might define two macros which allow the user to disable or enable  
**\DisableCrossrefs** the cross-referencing mechanism. Processing of files will be faster if only main index entries are generated (i.e., if **\DisableCrossrefs** is in force).

```
\def\DisableCrossrefs{\@bsphack\scan@allowedfalse\@esphack}
```

The macro **\EnableCrossrefs** will also disable any **\DisableCrossrefs** command encountered afterwards.

```
\def\EnableCrossrefs{\@bsphack\scan@allowedtrue
\def\DisableCrossrefs{\@bsphack\@esphack}\@esphack}
```

`\macro@switch` Now that we have the character which follows the escape character (in `\next`), we can determine whether it's a 'letter' (which category probably includes `@`). If it is, we let `\next` invoke a macro which assembles the full command name.

```
\def\macro@switch{\ifcat\noexpand\next a%
  \let\next\macro@name
```

Otherwise, we have a 'single-character' command name. For all such single-character names, we use `\short@macro` to process them into suitable index entries.

```
\else \let\next\short@macro \fi
```

Now that we know what macro to use to process the macro name, we invoke it ...

```
\next}
```

`\short@macro` This macro will be invoked (with a single character as parameter) when a single-character macro name has been spotted whilst scanning within the macrocode environment.

First we take a look at the `\index@excludelist` to see whether this macro name should produce an index entry. This is done by the `\ifnot@excluded` macro which assumes that the macro name is saved in `\macro@namepart`. Since the argument might be an active character, `\string` is used to normalize it.

```
\def\short@macro#1{\edef\macro@namepart{\string#1}%
  \ifnot@excluded
```

If necessary the index entry is produced by the macro `\produce@index`. Depending on the actual character seen, this macro has to do different things, so we pass the character as an argument.

```
\produce@index{#1}\fi
```

Then we disable the cross-referencing mechanism with `\scan@allowedfalse` and print the actual character. The index entry was generated first to ensure that no page break intervenes (recall that a `^^M` will start a new line).

```
\scan@allowedfalse#1%
```

After typesetting the character we can safely enable the cross-referencing feature again. Note that this macro won't be called (since `\macro@switch` won't be called) if cross-referencing is globally disabled.

```
\scan@allowedtrue }
```

`\produce@index` This macro is supposed to generate a suitable `\SortIndex` command for a given single-character control sequence. We test first for the cases which involve active characters (i.e. the backslash, the special escape character (if any), the space and the `^^M`). Using the `\if` test (testing for character codes), we have to ensure that the argument isn't expanded.

```
\def\produce@index#1{%
  \if\noexpand#1\special@escape@char
```

If the character is the special escape character (or the '`\`' in case there was none) the `\it@is@a` macro is used to produce the actual `\SortIndex` call.

```
\scan@allowedfalse \it@is@a\special@escape@char \else
```

Next comes the test for a '`\`' which must be the `\_13` expanding to `\bslash`.

```
\if\noexpand#1\bslash \it@is@a\bslash \else
```

Another possibility is `\_13`. Recall that `\space` produces a `\_10`.

```
\if\noexpand#1\space \it@is@a\space \else
```

The last<sup>16</sup> possibility of an active character is `^^M`. In this case we don't test for character codes, since it is easier to look if the character is equal to `\par`. (We are inside the macrocode environment.)

```
\ifx#1\par
```

If we end up here we have just scanned a `^^M` or something similar. Since this will be treated like `\_` by `TEX` we produce a corresponding index entry.

```
\it@is@a\space \else
```

The next three branches are needed because of bugs in our `makeindex` program. You can't produce unbalanced index entries<sup>17</sup> and you have to double a percent character. To get around these restrictions we use special macros to produce the `\index` calls.<sup>18</sup>

```
\if\noexpand#1\bgroup \LeftBraceIndex \else
\if\noexpand#1\egroup \RightBraceIndex \else
\if\noexpand#1\percentchar \PercentIndex \else
```

All remaining characters are used directly to produce their index entries. This is possible even for the characters which have special meanings in the index program, provided we quote the characters. (This is correctly done in `\it@is@a`.)

```
\it@is@a{\string#1}%
```

We now need a whole pile of `\fis` to match up with the `\ifs`.

```
\fi \fi \fi \fi \fi \fi \fi
```

`\macro@name` We now come to the macro which assembles command names which consist of one or more 'letters' (which might well include `@` symbols, or anything else which has a `\catcode` of 11).

To do this we add the letter to the existing definition of `\macro@namepart` (which you will recall was originally set to `@empty`).

```
\def\macro@name#1{\edef\macro@namepart{\macro@namepart#1}%
```

Then we grab hold of the *next* single character and let `\more@macroname` determine whether it belongs to the letter string forming the command name or is a non-letter.

```
\futurelet\next\more@macroname}
```

`\more@macroname` This causes another call of `\macro@name` to add in the next character, if it is indeed a letter.

```
\def\more@macroname{\ifcat\noexpand\next a%
\let\next\macro@name
```

Otherwise, it finishes off the index entry by invoking `\macro@finish`.

```
\else \let\next\macro@finish \fi
```

Here's where we invoke whatever macro was `\let` equal to `\next`.

```
\next}
```

<sup>16</sup> Well, it isn't the last active character after all. I added `\@noligs` some days ago and now `'` too is active. So we have to make sure that such characters don't get expanded in the index.

<sup>17</sup> This is possible for `TEX` if you use `{12}` or `}12}`, but `makeindex` will complain.

<sup>18</sup> Brian HAMILTON KELLY has written fixes for all three bugs. When they've found their way through all installations, the lines above will be removed. See page 265 if you already have them.

`\macro@finish` When we've assembled the full letter-string which forms the command name, we set the characters forming the entire command name, and generate an appropriate `\index` command (provided the command name is not on the list of exclusions). The '`\`' is already typeset; therefore we only have to output all letters saved in `\macro@namepart`.

```
\def\macro@finish{%
  \macro@namepart
```

Then we call `\ifnot@excluded` to decide whether we have to produce an index entry. The construction with `\@tempa` is needed because we want the expansion of `\macro@namepart` in the `\index` command.<sup>19</sup>

```
\ifnot@excluded
  \edef\@tempa{\noexpand\SpecialIndex{\backslash\macro@namepart}}%
  \@tempa \fi}
```

### 3.6 The index exclude list

The internal form of the index exclude list is

```
\@elt <macro name> \@elt <macro name> \@elt <macro name> ...
```

where `<macro name>` is a macro name like `\@tempa`. To test if a given macro name is on the list we only have to assign `\@elt` a proper meaning and then call `\index@excludelist`. This is faster than looping through the list and looking for the last element.

`\DoNotIndex` This macro is used to suppress macro names in the index. It starts off with a new group because we have to change the `\catcodes` of all characters which belong to 'letters' while macros are defined.

```
\def\DoNotIndex{\begingroup \MakePrivateLetters
```

Then we call the macro which actually reads the argument given by the user.

```
\do@not@index}
```

`\do@not@index` We make the `\do@not@index` macro `\long` since the user might want to exclude the `\par` macro.<sup>20</sup>

```
\long\def\do@not@index#1{%
```

Now we have to face the problem that `\index@excludelist` should be changed only locally (and we are already in a new group). Therefore we pass its contents to `\@gtempa`.

```
\global\let\@gtempa\index@excludelist
```

Then we define `\@elt` to allow expanding `\@gtempa` without damaging its contents.

```
\def\@elt{\noexpand\@elt\noexpand}%
```

The argument to `\do@not@index` is a set of macros separated by commas, so we use `\@for` to extract individual entries. Since `\@for` expands its argument, we hide it in another macro.<sup>21</sup>

```
\def\@tempa{#1}\@for\@tempb:=\@tempa\do
```

Now we can safely add new entries to `\@gtempa` (i.e. `\index@excludelist`).

```
{\xdef\@gtempa{\@gtempa \expandafter \@elt \@tempb}}%
```

<sup>19</sup> The `\index` command will expand its argument in the `\output` routine. At this time `\macro@namepart` might have a new value.

<sup>20</sup> Actually this doesn't work either because the argument is passed to `\@forloop` which isn't a `\long` macro. This will be fixed in a later version.

<sup>21</sup> Instead of using `\@for` one can make the comma active (expanding into `\@elt`) and then simply putting `\@gtempa` and `,#1` together. This will probably change.



After this we close the group and assign the retained value of `\@gtempa` to `\index@excludelist`.

```
\endgroup \let\index@excludelist\@gtempa}
```

`\index@excludelist` To get things going we have to initialize `\index@excludelist` and fix a bug in the `\@for` L<sup>A</sup>T<sub>E</sub>X `\@for` macro.

```
\def\index@excludelist{}
```

In the original `\@for` macro, `\@fortmp` gets its value using `\edef`. This means that it bombs if the contents of the second argument (i.e. the list A,B,C,...) contains undefined macros. Since this is possible if we use the `doc.sty` file to document a macro package which isn't loaded, we change the definition a little bit. Now it is tested only if the second argument expands to 'empty' when it is expanded once.<sup>22</sup>

```
\def\@for#1:=#2\do#3{\expandafter\def\expandafter\@fortmp\expandafter{#2}%
\ifx\@fortmp\@empty \else
\expandafter\@forloop#2,\@nil,\@nil,\@#1{#3}\fi}
```

`\ifnot@excluded` Now we take a look at the `\index@excludelist` to see whether a macro name saved in `\macro@namepart` should produce an index entry. This macro is a pseudo `\if`; it should expand to `\iftrue` or `\iffalse` depending on the contents of `\index@excludelist`. We use `\if@tempswa` for this purpose and initialize it with `true`.

```
\def\ifnot@excluded{\@tempswatrue
```

Then we `\let` the macro `\@elt` equal to a test macro (which is supposed to change the switch if the `\macro@namepart` is on the list) and call `\index@excludelist`. This is all done in an `\hbox` so any garbage produced by calling `\index@includelist` will vanish. The test macro uses `\aftergroup` to avoid global changes while communicating with the outside world.

```
\setbox\z@\hbox{\let\@elt\exclude@test \index@excludelist}}%
```

Finally we call `\if@tempswa`.

```
\if@tempswa}
```

Note however that since we have called `\if@tempswa` inside this macro, such a construction can't be used inside a conditional at the same expansion level [3, p211].

`\exclude@test` Strictly speaking, `\macro@namepart` contains only the name without the backslash. So we use `\expandafter` and `\csname` to produce the actual macro name (arguments to `\ifx` are not expanded).

```
\def\exclude@test#1{%
\expandafter \ifx \csname\macro@namepart\endcsname #1%
```

The `\ifx` test will be true if the argument and the constructed macro are the same. In this case we have to change the switch. We also change the definition of `\@elt` because our goal is reached and we can gobble up the tail of the list. As mentioned above, switch changing is deferred until after the current group by using `\aftergroup`.

```
\aftergroup\@tempswafalse \let\@elt\@gobble \fi}
```

<sup>22</sup> This is exactly the way in which the argument is used in the second part of the definition (in the actual loop). Therefore I am inclined to call it a bug and not a feature.

### 3.7 Macros for generating index entries

Here we provide default definitions for the macros invoked to create index entries; these are either invoked explicitly, or automatically by `\scan@macro`. As already mentioned, the definitions given here presuppose that the `.idx` file will be processed by Chen's `makeindex` program — they may be redefined for use with the user's favourite such program.

To assist the reader in locating items in the index, all such entries are sorted alphabetically *ignoring* the initial `\`; this is achieved by issuing an `\index` command which contains the 'actual' operator for `makeindex`. The default value for the latter operator is `@`, but the latter character is so popular in L<sup>A</sup>T<sub>E</sub>X style files that it is necessary to substitute another character. This is indicated to `makeindex` by means of an 'index style file'<sup>23</sup>; the character selected for this function is `=`, and therefore this character too must be specially treated when it is met in a T<sub>E</sub>X command.

<code>\actualchar</code>	First come the definitions of <code>\actualchar</code> , <code>\quotechar</code> and <code>\levelchar</code> . Note, that our defaults are not the ones used by the <code>makeindex</code> program without a style file.
<code>\quotechar</code>	
<code>\levelchar</code>	<pre> @ifundefined{actualchar}{\def\actualchar{=}}{} @ifundefined{quotechar}{\def\quotechar{!}}{} @ifundefined{levelchar}{\def\levelchar{&gt;}}{} </pre>
<code>\encapchar</code>	The <code>makeindex</code> default for the <code>\encapchar</code> isn't changed. <pre> @ifundefined{encapchar}{\def\encapchar{!}}{} </pre>
<code>\verbatimchar</code>	We also need a special character to be used as a delimiter for the <code>\verb*</code> command used in the definitions below. <pre> @ifundefined{verbatimchar}{\def\verbatimchar{+}}{} </pre>
<code>\SpecialIndex</code>	<p>The <code>\SpecialIndex</code> command creates index entries for macros. If the argument is <code>xyz</code>, the command produces <code>\indexentry{xyz=\verb!*+xyz+}{n}</code> given the above defined defaults for <code>\actualchar</code>, <code>\quotechar</code> and <code>\verbatimchar</code>. We first remove the initial <code>\</code> to get a better index.</p> <pre> \def\SpecialIndex#1{\@bsphack\index{\expandafter\@gobble\string#1\actualchar </pre> <p>Then follows the actual entry. A <code>\quotechar</code> is placed before the <code>*</code> to allow its use as a special <code>makeindex</code> character. Again <code>\@bsphack</code> and <code>\@esphack</code> are used to make the macros invisible.</p> <pre> \string\verb\quotechar*\verbatimchar\string#1\verbatimchar}\@esphack} </pre>
<code>\SpecialMainIndex</code>	<p>The <code>\SpecialMainIndex</code> macro is used to cross-reference the names introduced by the macro environment. The action is as for <code>\SpecialIndex</code>, except that <code>makeindex</code> is instructed to 'encapsulate' the entry with the string <code> main</code> to cause it to generate a call of the <code>\main</code> macro.</p> <pre> \def\SpecialMainIndex#1{\@bsphack\index{\expandafter\@gobble \string#1\actualchar \string\verb \quotechar*\verbatimchar \string#1\verbatimchar \encapchar main}% \@esphack} </pre> <p>The <code>\SpecialUsageIndex</code> is literally the same—only we use <code>usage</code> instead of <code>main</code>.</p> <pre> \def\SpecialUsageIndex#1{\@bsphack\index{\expandafter\@gobble\string#1% \actualchar\string\verb\quotechar*\verbatimchar\string#1\verbatimchar \encapchar usage}\@esphack} </pre>
<code>\SpecialUsageIndex</code>	

<sup>23</sup> A file suitable to this task is provided amongst the supporting files for this style file in `gind.ist`.

`\SpecialEnvIndex` Indexing environments is done a little bit differently; we produce two index entries with the `\SpecialEnvIndex` macro:

```
\def\SpecialEnvIndex#1{\@bsphack
```

First we sort the environment under its own name stating in the actual entry that this is an environment.

```
\index{#1\actualchar{\tt #1} (environment)\encapchar usage}%
```

The second entry is sorted as a subitem under the key 'environments':

```
\index{environments:\levelchar{\tt #1}\encapchar usage}\@esphack}
```

Because both entries corresponds to 'descriptions' of the environment, we encapsulate the page numbers with the `\usage` macro.

`\SortIndex` This macro is used to generate the index entries for any single-character command that `\scan@macro` encounters. The first parameter specifies the lexical order for the character, whilst the second gives the actual characters to be printed in the entry. It can also be used directly to generate index entries which differ in sort key and actual entry.

```
\def\SortIndex#1#2{\index{#1\actualchar#2}}
```

`\it@is@a` This macro is supposed to produce a correct `\SortIndex` entry for a given character. Since this character might be recognised as a 'command' character by the index program used, all characters are quoted with the `\quotechar`.

```
\def\it@is@a#1{\SortIndex{\quotechar #1}%
                          {\string\verb\quotechar*\verbatimchar
                          \quotechar\backslash\quotechar#1\verbatimchar}}
```

`\LeftBraceIndex` `\RightBraceIndex` These two macros fix the problems with `makeindex`. Note the 'hack' with `\iffalse}\fi` to satisfy both `TeX` and the `makeindex` program. When this is written to the `.idx` file `TeX` will see both braces (so we get a balanced text). `makeindex` will also see balanced braces but when the actual index entry is again processed by `TeX` the brace in between `\iffalse \fi` will vanish.

```
\@ifundefined{LeftBraceIndex}{\def\LeftBraceIndex{%
  \index{\bgroup\actualchar\string\verb\quotechar*\verbatimchar
        \quotechar\backslash{\verbatimchar\string\iffalse}\string\fi}}{}
\@ifundefined{RightBraceIndex}{\def\RightBraceIndex{%
  \index{\egroup\actualchar\string\iffalse{\string\fi}\string\verb
        \quotechar*\verbatimchar\quotechar\backslash\verbatimchar}}{}
\@ifundefined{PercentIndex}{\def\PercentIndex{%
  \index{\quotechar\percentchar\actualchar\string\verb
        \quotechar*\verbatimchar\quotechar\backslash
        \percentchar\percentchar\verbatimchar}}{}
\catcode'\%=12 \gdef\percentchar{}}
\def\PercentIndex{\it@is@a\percentchar}
\typeout{The doc style assumes that a \percentchar\space
will be processed as a \percentchar\space by the index program!}
```

`\PercentIndex` Here is one solution for the percent bug in `makeindex`. The macro `\percentchar` denotes a `%12`.

```
\@ifundefined{PercentIndex}{\def\PercentIndex{%
  \index{\quotechar\percentchar\actualchar\string\verb
        \quotechar*\verbatimchar\quotechar\backslash
        \percentchar\percentchar\verbatimchar}}{}
\catcode'\%=12 \gdef\percentchar{}}
\def\PercentIndex{\it@is@a\percentchar}
\typeout{The doc style assumes that a \percentchar\space
will be processed as a \percentchar\space by the index program!}
```

If you've got a newer `makeindex` program which handles the percents correctly you have to uncomment the next three lines.<sup>24</sup> Otherwise you will get `\%%` entries in your index.

```
% \def\PercentIndex{\it@is@a\percentchar}
% \typeout{The doc style assumes that a \percentchar\space
% will be processed as a \percentchar\space by the index program!}
```

<sup>24</sup> This is the only change which is allowed in this file! All other changes should be made through a style file, predefining internal quantities.

### 3.8 Redefining the index environment

The index is set in three columns, and will start on the same page as, and underneath, the last part of the text of the documented style file, if possible. The last page will be reformatted with balanced columns. We make use of the multicols environment which is described elsewhere (in an article scheduled to appear in the next issue of TUGboat).

```

ATEX
counter declared above to denote the number of columns and insert the 'index prologue'
text (which might contain a \section call, etc.). See the default definition for an
example.

\def\theindex{\multicols\c@IndexColumns[\index@prologue][\IndexMin]%

Then we make a few last minute assignments to read the individual index \items and
finish off by ignoring any initial space.

\IndexParms \let\item\@idxitem \ignorespaces}

\endtheindex At the end of the index, we have only to end the multicols environment.

\let\endtheindex=\endmulticols

\IndexPrologue \index@prologue The \IndexPrologue macro is used to place a short message into the document above
the index. It is implemented by redefining \index@prologue, a macro which holds
the default text. We'd better make it a \long macro to allow \par commands in its
argument.

\long\def\IndexPrologue#1{\@bsphack\def\index@prologue{#1}\@esphack}

Now we test whether the default is already defined by another style file. If not we
define it.

\ifundefined{index@prologue}
{\def\index@prologue{\section*{Index}%
\markboth{Index}{Index}%
The italic numbers denote the pages where the
corresponding entry is described,
numbers underlined point to the definition,
all others indicate the places where it is used.
}}{}

\IndexParms These are some last-minute assignments for formatting the index entries. They are
defined in a separate macro so that a user can substitute different definitions. We start
by defining the various parameters controlling leading and the separation between the
two columns. The entire index is set in \small size.

\ifundefined{IndexParms}
{\def\IndexParms{%
\parindent \z@
\columnsep 15pt
\parskip 0pt plus 1pt
\righskip 15pt

```

```

\mathsurround \z@
\parfillskip=-15pt
\small

\@idxitem Index items are formatted with hanging indentation for any items which may require
\subitem more than one line.
\subsubitem \def\@idxitem{\par\hangindent 30pt}%
Any sub-item in the index is formatted with a 15pt indentation under its main heading.
\def\subitem{\@idxitem\hspace*{15pt}}%
Whilst sub-sub-items go in a further 10pt.
\def\subsubitem{\@idxitem\hspace*{25pt}}%

\indexspace The makeindex program generates an \indexspace before each new alphabetic section
commences. After this final definition we end the \@ifundefined and the definition
of \IndexParms.
\def\indexspace{\par\vspace{10pt plus 2pt minus 3pt}}%
}}{}

\efill This definition of \efill is intended to be used after index items which have no
following text (for example, "see" entries). It just ensures that the current line is
filled, preventing "Underfull \hbox" messages.
\def\efill{\hfill\nopagebreak}%

\pfill The following definitions provide the \pfill command; if this is specified in the index
\dotfil style file to makeindex as the delimiter to appear after index items, then the intervening
\dotfill space before the referenced page numbers will be filled with dots, with a little white
space interpolated at each end of the dots. If the line is broken the dots will show up
on both lines.
\def\dotfill{\leaders\hbox to.6em{\hss .\hss}\hskip\z@ plus 1fill}%
\def\dotfil{\leaders\hbox to.6em{\hss .\hss}\hfil}%
\def\pfill{\unskip~\dotfill\penalty500\strut\nobreak
\dotfil~\ignorespaces}%

\* Here is the definition for the \* macro. It isn't used in this set of macros.
\def*\{\leavevmode\lower.8ex\hbox{\$, \widetilde{\ } \}, \$}}

\main The defining entry for a macro name is flagged with the string |main25 in the \index
command; makeindex processes this so that the \main macro will be invoked to underline
the page number(s) on which the definition of the macro will be found.
\@ifundefined{main}{\def\main#1{\underline{#1}}}{}}

\usage The \usage macro is used to indicate entries describing the usage of a macro. The
corresponding page number(s) will be set in italics.
\@ifundefined{usage}{\def\usage#1{{\it #1}}}{}}

\printindex To read in and print the sorted index, just put the \printindex command as the last
(commented-out, and thus executed during the documentation pass through the file)
command in your style file. Precede it by any bibliography commands necessary for
your citations.
Alternatively, it may be more convenient to put all such calls amongst the arguments
of the \StopEventually macro, in which case a \Finale command should appear at
the end of your file.
\def\printindex{\@input{jobname.ind}}

```

<sup>25</sup> With the current definition of \encapchar substituted for |

### 3.9 Dealing with the change history<sup>26</sup>

To provide a change history log, the `\changes` command has been introduced. This takes three arguments, respectively, the version number of the file, the date of the change, and some detail regarding what change has been made. The first of these arguments is otherwise ignored, but the others are written out and may be used to generate a history of changes, to be printed at the end of the document. However, note that Chen's standard `makeindex` program limits any textual field to just 64 characters; therefore, it is important that the number of characters in the second and third parameters should not exceed 61 altogether (to allow for the parentheses placed around the date).

`\changes` The output of the `\changes` command goes into the  $\langle Glossary\_File \rangle$  and therefore uses the normal `\indexentry` commands. Thus `makeindex` or a similar program can be used to process the output into a sorted "glossary". The `\changes` command commences by taking the usual measures to hide its spacing, and then redefines `\protect` for use within the argument of the generated `\indexentry` command.

```
\def\changes#1#2#3{\@bsphack{%
  \def\protect##1{\string##1\space}%
```

We now create the requisite `\glossary` command, and output it.

```
\edef\@tempa{\noexpand\glossary{\expandafter\@gobble
  \saved@macroname\actualchar
  \string\verb\quotechar*%
  \verbatimchar\saved@macroname
  \verbatimchar\levelchar#2%
  \actualchar(#2) #3}}%
\@tempa}\@esphack}
```

`\saved@macroname` The entries are sorted for convenience by the name of the most recently introduced macroname (i.e., that in the most recent `\begin{macro}` command). We therefore provide `\saved@macroname` to record that argument, and provide a default definition in case `\changes` is used outside a macro environment. (This is a *wicked* hack to get such entries at the beginning of the sorted list!)

```
\def\saved@macroname{ ' General Changes ' }
```

`\RecordChanges` To cause the changes to be written (to a `.glo`) file, we define `\RecordChanges` to invoke L<sup>A</sup>T<sub>E</sub>X's usual `\makeglossary` command.

```
\let\RecordChanges\makeglossary
```

`\GlossaryMin` The remaining macros are all analogues of those used for the `theindex` environment.  
`\c@GlossaryColumns` When the glossary is started we compute the space which remains at the bottom of the current page; if this is greater than `\GlossaryMin` then the first part of the glossary will be placed in the available space. The number of columns set are controlled by the counter `\c@GlossaryColumns` which can be changed with a `\setcounter` declaration.

```
\newdimen\GlossaryMin      \GlossaryMin      = 80pt
\newcount\c@GlossaryColumns \c@GlossaryColumns = 2
```

`\theglossary` We start with a few last minute assignments to read the individual glossary `\items`. Note the `\par` at the beginning. If we leave it out, parameter changes done by the `\GlossaryParms` macro might affect the paragraph above the glossary.

```
\def\theglossary{\par \GlossaryParms \let\item\@idxitem
```

<sup>26</sup> The whole section was proposed by Brian HAMILTON KELLY. He also documented and debugged the macros as well as many other parts of this style option.

Now we start the multi-column mechanism. We use `\c@GlossaryColumns` to denote the number of columns and insert the 'glossary prologue' text which might contain a `\section` call etc. See the default definition for an example.

```
\multicols\c@GlossaryColumns[\glossary@prologue][\GlossaryMin]}
```

`\endglossary` At the end of the glossary, we've only to end the multicols environment.

```
\let\endtheglossary=\endmulticols
```

`\GlossaryPrologue` The `\GlossaryPrologue` macro is used to place a short message above the glossary into the document. It is implemented by redefining `\glossary@prologue`, a macro which holds the default text. We better make it a long macro to allow `\par` commands in its argument.

`\glossary@prologue`

```
\long\def\GlossaryPrologue#1{\@bsphack
\def\glossary@prologue{#1}%
\@esphack}
```

Now we test whether the default is already defined by another style file. If not we define it.

```
\ifundefined{glossary@prologue}
{\def\glossary@prologue{\begingroup\parfillskip=0pt plus 1fil
\section*{\Change History}\par
\endgroup
\markboth{\Change History}{\Change History}%
}}{}
```

`\GlossaryParms` Unless the user specifies otherwise, we set the change history using the same parameters as for the index.

```
\ifundefined{GlossaryParms}{\let\GlossaryParms=\IndexParms}{}
```

`\PrintChanges` To read in and print the sorted change history, just put the `\PrintChanges` command as the last (commented-out, and thus executed during the documentation pass through the file) command in your style file. Alternatively, this command may form one of the arguments of the `\StopEventually` command, although a change history is probably *not* required if only the description is being printed.

The command assumes that `makeindex` or some other program has processed the `.gls` file to generate a sorted `.gls` file.

```
\def\PrintChanges{\@input{\jobname.gls}}
```

### 3.10 Bells and whistles

`\StopEventually` Here is the default definition for `\StopEventually`, we simply save its argument in the macro `\Finale`.

`\Finale`

`\OnlyDescription`

```
\long\def\StopEventually#1{\@bsphack\def\Finale{#1}\@esphack}
```

When the user places an `\OnlyDescription` declaration in the driver file the document should only be typeset up to `\StopEventually`. We therefore have to redefine this macro.

```
\def\OnlyDescription{\@bsphack\long\def\StopEventually##1{%
```

In this case the argument of `\StopEventually` should be set and afterwards `TEX` should stop reading from this file. Therefore we finish this macro with

```
##1\endinput}\@esphack}
```

`\meta` The `\meta` macro is very elementary.

```
\def\meta#1{\mbox{$\langle$it#1/$\rangle$}}
```

`\IndexListing` This next macro may be used to read in a separate file (possibly a style file that is *not* documented by this means) and set it verbatim, whilst scanning for macro names and indexing the latter. This could be a useful first pass in preparing to generate documentation for the file read.

```
\def\IndexListing#1{%
```

We commence by setting up a group, and initializing a `\trivlist` as is normally done by a `\begin{macrocode}` command.

```
\begingroup \macro@code
```

We also make spacing behave as in the macrocode environment, because otherwise all the spaces will be shown explicitly.

```
\frenchspacing \@vobeyspaces
```

Then it only remains to read in the specified file, and finish off the `\trivlist`.

```
\input{#1}\endtrivlist
```

Of course, we need to finish off the group as well.

```
\endgroup}
```

`\maketitle` The macro to generate titles is easily altered in order that it can be used more than once (an article with many titles). In the original, diverse macros were concealed after use with `\relax`. We must cancel anything that may have been put into `\@thanks`, etc., otherwise *all* titles will carry forward any earlier such setting!

```
\def\maketitle{\par
  \begingroup \def \thefootnote {\fnsymbol {footnote}}%
  \setcounter {footnote}\z@
  \def \@makefnmark {\hbox to \z@{${\@thefnmark }$}\hss }}%
  \if@twocolumn \twocolumn [\@maketitle ]
  \else \global \@topnum \z@ \@maketitle \fi
  \@thanks \endgroup
```

If the driver file documents many files, we don't want parts of a title of one to propagate to the next, so we have to cancel these:

```
\setcounter {footnote}\z@
\gdef\@date{\today}\gdef\@thanks{}%
\gdef\@author{}\gdef\@title{}}
```

### 3.11 Layout parameters for documenting style files

`\tolerance` People documenting style files would probably rather have things "sticking out" in overfull `\hboxes` and poorish spacing, because they probably don't want to spend a lot of time on making all the line breaks perfect!

```
\tolerance=1000\relax
```

The following `\mathcode` definitions allow the characters `'\'` and `'@'` to appear in `\tt` font when invoked in math mode;<sup>27</sup> particularly for something like `\@abc = 1`.

If an *old* version of the german style option is in force, then the `"` character is active and would upset the definition of the *(16-bit number)* quantities below, therefore we change the `\catcode` of `"` inside a group, and use `\global`.

```
{ \catcode'"=12
  \global\mathcode'\="705C \global\mathcode'@"="7040 }
```

<sup>27</sup> You may wonder why the definitions state that both characters belong to the *variable family* (i.e. the number 7 in front). The reason is this: Originally the `\mathcode` of `\` was defined to be `"075C`, i.e. ordinary character number 92 (hex 5C) in math family number 7 which is the typewriter family in standard L<sup>A</sup>T<sub>E</sub>X. But this file should not depend on this specific setting, so I changed these `\mathcode`s to work with any family assignments. For an example see the article about the new font selection scheme.



**\DocstyleParms** This macro can be used, for example, to assign new values to `\MacrocodeTopsep` and `\MacroIndent` and some other internal registers. If it is already defined, the default definition won't be carried out. Note that it is necessary to assign new values via this macro if it should be done in a style file (like `ltugbot.sty` for example) since the registers are undefined before `doc.sty` is read in. The default values for the internal registers are scattered over this file. Here we only execute `\makeindex` because this declaration can't be overwritten otherwise.

```
\@ifundefined{DocstyleParms}{\makeindex}{}

```

Now we allow overwriting the values by calling `\DocstyleParms`.

```
\DocstyleParms \let\DocstyleParms\relax

```

**\AmSTeX** Here are a few definitions which can usefully be employed when documenting style files: now we can readily refer to `\AASTeX`, `\BIBTeX` and `\SLiTeX`, as well as the usual `TeX` and `LaTeX`.

```
\@ifundefined{AmSTeX}
  {\def\AmSTeX{\leavevmode\hbox{${\cal A}\kern-.2em\lower.376ex%
    \hbox{${\cal M}}\kern-.2em\cal S$-\TeX}}}{ }
\@ifundefined{BibTeX}
  {\def\BibTeX{{\rm B}\kern-.05em{\sc i}\kern-.025em b}\kern-.08em%
    T\kern-.1667em\lower.7ex\hbox{E}}\kern-.125emX}}{ }
\@ifundefined{SliTeX}
  {\def\SliTeX{{\rm S}\kern-.06em{\smc l}\kern-.035emi}\kern-.06em\TeX}}{ }

```

**\PlainTeX** There's even a `PLAIN TeX` and a `WEB`.

```
\Web
\@ifundefined{PlainTeX}{\def\PlainTeX{{\sc Plain}\kern2pt\TeX}}{ }
\@ifundefined{Web}{\def\Web{{\sc Web}}}{ }

```

### 3.12 Changing the `\catcode` of `%`

**\MakePercentIgnore** And finally the most important bit: we change the `\catcode` of `'%` so that it is ignored (which is how we are able to produce this document!). We provide two commands to do the actual switching. Then `\MakePercentIgnore` is called as the last command in this file.

```
\def\MakePercentIgnore{\catcode'\%9\relax}
\def\MakePercentComment{\catcode'\%14\relax}

```

### References

- [1] G. A. BÜRGER. *Wunderbare Reisen zu Wasser und zu Lande, Feldzüge und lustige Abenteuer des Freyherrn v. Münchhausen*. London, 1786 & 1788.
- [2] D. E. KNUTH. *Literate Programming*. *Computer Journal*, Vol. 27, pp. 97–111, May 1984.
- [3] D. E. KNUTH. *Computers & Typesetting (The TeXbook)*. Addison-Wesley, Vol. A, 1986.
- [4] L. LAMPORT. *MakeIndex: An Index Processor for LaTeX*. 17 February 1987. (Taken from the file `makeindex.tex` provided with the program source code.)
- [5] R. E. RASPE (\*1737, †1797). *Baron Münchhausens narrative of his marvellous travels and campaigns in Russia*. Oxford, 1785.

## Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

- Symbols**
- `\*` ..... 249, 267
  - `\@for` ..... 262, 263
  - `\@idxitem` .....  
..... 249, 266, 267, 268
  - `\@verbatim` ... 254, 254
  - `^^A` ..... 246, 251
- A**
- `\active@escape@char`  
..... 257, 258, 259
  - `\actualchar` ... 248,  
264, 264, 265, 268
  - `\AmSTeX` ..... 271
- B**
- `\BibTeX` ..... 271
  - `\blank@linefalse` ..  
..... 252, 252
  - `\blank@linetrue` 252, 252
  - `\bslash` ..... 254,  
258, 260, 262, 265
- C**
- `\c@GlossaryColumns` .  
..... 268, 269
  - `\c@IndexColumns` 266, 266
  - `\changes` ..... 250, 268
  - `\check@percent` 255, 255
  - `\close@crossref` 252, 259
  - `\columnsep` ... 249, 266
- D**
- `\DescribeEnv` .. 247, 253
  - `\DescribeMacro` 247, 253
  - `\DisableCrossrefs` .  
..... 248, 259
  - `\do@not@index` . 262, 262
  - `\docdate` ..... 251, 251
  - `\DocstyleParms` 250, 271
  - `\DoNotIndex` 248, 256, 262
  - `\dotfil` ..... 267
  - `\dotfill` ..... 267
- E**
- `\efill` ..... 267
  - `\EnableCrossrefs` ..  
..... 248, 259
  - `\encapchar` ... 248,  
264, 264, 264, 265
  - `\endglossary` ..... 269
  - `\endmacro` ..... 256
  - `\endmacrocode` ..... 252
  - `\endmacrocode*` ..... 253
- `\endmulticols` ..... 269
  - `\endtheglossary` ... 269
  - `\endtheindex` ..... 266
- environments:
- macrocode\* ..... 247
  - macrocode ..... 246
  - macro ..... 247
  - theindex ..... 249
  - verbatim\* ..... 247
  - verbatim ..... 247
- `\exclude@test` ..... 263
- F**
- `\filedate` .... 251, 251
  - `\fileversion` .. 251, 251
  - `\Finale` ..... 250, 269
- G**
- `\glossary@prologue` .  
..... 269, 269
  - `\GlossaryMin` .. 268, 269
  - `\GlossaryParms` 268, 269
  - `\GlossaryPrologue` . 269
- I**
- `\ifblank@line` ..... 252
  - `\ifnot@excluded` ...  
..... 260, 262, 263
  - `\ifscan@allowed` ... 259
  - `\index@excludelist` .  
..... 262, 263, 263, 263
  - `\index@prologue` 266, 266
  - `\IndexListing` . 250, 270
  - `\IndexMin` .....  
..... 249, 250, 266, 266
  - `\IndexParms` 249, 266, 266
  - `\IndexPrologue` 249, 266
  - `\indexspace` ..... 267
  - `\init@crossref` 252, 258
  - `\it@is@a` 260, 261, 265, 265
- L**
- `\LeftBraceIndex` 261, 265
  - `\levelchar` .....  
..... 249, 264, 265, 268
- M**
- `\macro` ..... 255
  - macro (environment) . 247
  - `\macro@` ..... 255
  - `\macro@cnt` ... 255, 256
  - `\macro@code` .....  
..... 251, 251, 253, 270
  - `\macro@finish` ..... 262
- `\macro@name` 260, 261, 261
  - `\macro@namepart` ...  
..... 259, 260-263
  - `\macro@switch` . 259, 260
  - `\macrocode` ..... 251
  - macrocode (environment)  
..... 246
  - `\macrocode*` ..... 253
  - macrocode\* (environment)  
..... 247
  - `\MacrocodeTopsep` ..  
..... 247, 250, 251, 252
  - `\MacroFont` ... 247,  
252, 252, 254, 257
  - `\MacroIndent` .....  
..... 247, 250, 252, 252
  - `\MacroTopsep` .....  
..... 247, 250, 255, 257
  - `\main` ..... 250, 267
  - `\makeindex` ..... 250
  - `\MakePercentComment` 271
  - `\MakePercentIgnore` . 271
  - `\MakePrivateLetters`  
..... 255, 258, 259, 262
  - `\maketitle` ... 250, 270
  - `\marginparpush` 250, 254
  - `\marginparwidth` 250, 254
  - `\mathsurround` ..... 249
  - `\meta` ..... 250, 269
  - `\more@macroname` 261, 261
  - `\multicols` ... 266, 269
- N**
- `\newif` ..... 252, 259
- O**
- `\OnlyDescription` ..  
..... 250, 269
- P**
- `\par` ..... 252,  
254, 261, 267-270
  - `\parfillskip` ..... 249
  - `\parindent` ..... 249
  - `\parskip` ..... 249
  - `\percentchar` .. 261, 265
  - `\PercentIndex` . 261, 265
  - `\pfill` ..... 267
  - `\PlainTeX` ..... 271
  - `\predisplaypenalty` .  
..... 252, 254
  - `\PrintChanges` ..... 269
  - `\PrintDescribeEnv` .  
..... 247, 254, 257

<code>\PrintDescribeMacro</code>	<code>\scan@allowedtrue</code>	<b>T</b>
.... 247, 254, 257	.... 259, 259, 260	<code>\theglossary</code> ..... 268
<code>\printindex</code> ..... 267	<code>\scan@macro</code> ... 258, 259	<code>\theindex</code> ..... 266
<code>\PrintMacroName</code> ...	<code>\short@macro</code> .. 260, 260	<code>theindex</code> (environment)
.... 247, 256, 257	<code>\SlitEX</code> ..... 271	..... 249
<code>\produce@index</code> 260, 260	<code>\smc</code> ..... 271	<code>\tolerance</code> ... 250, 270
<b>Q</b>	<code>\SortIndex</code> 249, 265, 265	<b>U</b>
<code>\quotechar</code> .....	<code>\special@escape@char</code>	<code>\usage</code> ..... 250, 267
. 248, 264, 264, 268	.... 257, 258-260	<b>V</b>
<b>R</b>	<code>\SpecialEnvIndex</code> ..	<code>\verbatim</code> ..... 254
<code>\RecordChanges</code> .... 268	.... 249, 254, 265	<code>verbatim</code> (environment)
<code>\reversemarginpar</code> . 254	<code>\SpecialEscapechar</code> .	..... 247
<code>\RightBraceIndex</code> ..	. 248, 257, 258, 259	<code>verbatim*</code> (environment)
..... 261, 265	<code>\SpecialIndex</code> .....	..... 247
<code>\rightskip</code> ..... 249	.... 249, 262, 264	<code>\verbatimchar</code> . 249,
<b>S</b>	<code>\SpecialMainIndex</code> .	264, 264, 265, 268
<code>\saved@macroname</code> ..	<code>\SpecialUsageIndex</code> .	<b>W</b>
.... 255, 268, 268	.... 249, 254, 264	<code>\Web</code> ..... 271
<code>\scan@allowedfalse</code> .	<code>\StopEventually</code> 250, 269	<b>X</b>
.... 259, 259, 260	<code>\subitem</code> ..... 267	<code>\xmacro@code</code> .. 251, 253
	<code>\subsubitem</code> ..... 267	
	<code>\sxmacro@code</code> . 253, 253	

◊ Frank Mittelbach  
 Fachbereich Mathematik  
 Universitat Mainz  
 Staudinger Weg 9  
 D-6500 Mainz  
 Federal Republic of Germany  
 Bitnet: schoepf@dmznat51

---

## The autodoc-Option\*

B Hamilton Kelly

### Abstract

This style option is used as an adjunct to the doc style option, and facilitates the documentation of style and other files, by making it unnecessary to have a separate driver file for each file being documented.

---

### Contents

<b>1</b>	<b>Introduction</b>	<b>274</b>
<b>2</b>	<b>Description of the Macros</b>	<b>276</b>
2.1	Reading in additional style options . . . . .	276
2.1.1	Recognizing parts of a file specification . . . . .	277
2.2	Scanning the file names in <code>\docnames</code> . . . . .	278
<b>3</b>	<b>Processing the specified files</b>	<b>280</b>
3.1	Processing the file itself . . . . .	280
3.2	Handling toc files, etc . . . . .	282
<b>4</b>	<b>Reading in the doc Style Option</b>	<b>284</b>

---

### 1 Introduction

Frank Mittelbach's [1] excellent doc style option has one slight drawback: it is necessary to write a small L<sup>A</sup>T<sub>E</sub>X "driver" file for each file being documented. The `\documentstyle` used for this will always be `article`, and the last style option on the command will always be `doc`. However, because the documentation of a *style option* file should obviously include examples of the use of the commands which it provides, it is necessary also to include as a further option the name of the style option being documented, in order that its commands may be available as the documentation is generated.

The style option presented here removes the necessity for writing a separate driver file; it works by prompting the user for the name(s) of the file(s) being documented, then (optionally) reading in those files *and* the doc style file, so that the driver file, which may be common to all style options, just has to issue a single command to cause all the referenced style files to be processed. As each file is processed, this style file opens and closes various auxiliary files<sup>1</sup> appropriate to the file being documented; the names of these files are taken from `\docname`, which gets redefined as necessary, and, in fact, these macros also redefine `\jobname` as the documenting process progresses. Unfortunately, `autodoc` is unable to do anything about the fact that the `.dvi` and log files will have already been opened with the name of the driver file. For this it will be necessary to use the operating system's facilities for renaming files.

`\processdocs` This is the only macro that the user needs to know about; the driver file for use with the `autodoc` option should look something like this:

```
\documentstyle[autodoc]{article}
```

---

\* This is version v2.1f dated 30-Apr-1989

<sup>1</sup> `aux`, `idx`, `ind`, `toc`, etc.

```

\begin{document}
\processdocs
\end{document}

```

All the work and interaction with the user is performed when the `autodoc` option is first read in. At present, it firstly asks the user the question:

What file(s) are you documenting?

to which the user should respond with the name(s) of the file(s) to be documented. If there is more than one such file, the names should be separated by commas, not spaces.

If any of the files are `.sty` files, which are *not* `doc.sty` or `autodoc.sty`, then the user is further prompted for each file to specify whether the file shall be read as an *option*, with:

Does the description of `<filename>` use its macros?

The user should answer `yes`<sup>2</sup> or `no`. If an affirmative answer is given, the user is further prompted:

Can reading of `<filename>` be deferred until it is processed?

It will be found that most style option files don't actually use any commands which may only be executed in the preamble, so stack save space can be conserved by postponing the reading of the file until just before it is processed. However, style options which *do* utilize such commands must be read now, before `doc.sty` is input.

For producing a summary of all style files, etc., available at a site, the user may want to typeset just the descriptions of the files. Therefore, he is prompted:

Should `<filename>` be fully documented?

A negative response will lead to that file being processed by `doc` with `\OnlyDescription` in force.

Once a file has been fully cross-referenced, it's pointless to keep on processing it with `\EnableCrossrefs` in effect, so the user is given the option of suppressing this for each file; the user is prompted with:

Should `<filename>` be cross referenced?

Having decided what files are to be documented, and how, the user is given the option of reading in additional style options *before* the `doc` option is read; this permits inclusion of such options as `german.sty`. The user prompt is:

Give the names of any further options (WITHOUT `.sty`):

If the user doesn't wish to use any such additional options, a response of just pressing the RETURN key will terminate user interaction; otherwise, the specified options will all be read.

After user interaction is completed, the specified files will be read and their documentation produced. Each file will generate its own auxiliary files (`.idx`, `.toc`, etc.), including an `.aux` file; it will be necessary to make the customary two or three passes through the files to complete all cross-references, indices, etc. The recommended sequence is:

1. LATEX, to generate the first `.idx` and `.toc` files.
2. `makeindex`, to generate an `.ind` file (which will have the wrong page numbers, since the `.toc` file has yet to be read).

---

<sup>2</sup> In fact, any response starting with the letter 'Y' or 'y' is interpreted as 'yes, and anything else as 'no.'

3. LATEX, to regenerate the auxiliary files, this time with correct page references in them.
4. makeindex, to generate a correct .ind file.
5. LATEX, which hopefully will have correct page numbers throughout.

## 2 Description of the Macros

As with all style options, we commence by identifying ourselves on the terminal and in the log file:

```
\typeout{Style-Option: 'autodoc' \fileversion\space\space
          \filedate\space (BHK)}
```

**\docnames** This style file is to be used (in conjunction with the doc style option) for documenting style *option* files (including itself), (main) style files, and even any other L<sup>A</sup>T<sub>E</sub>X document.

Our first action, therefore, is to prompt the user for the name(s) of the file(s) being documented.

```
\typein[\docnames]{What style file(s) are you documenting? }
```

Let's allow the user the luxury of not having to remember to type in the correct case...

```
\edef\next{\def\noexpand\docnames{\docnames}}
\lowercase\expandafter{\next}
```

### 2.1 Reading in additional style options

**\styleoption** When we come to document each of the files in `\docnames`, style option files need to be treated specially, because the description of such a file is likely to want to give examples of the facilities which it defines, so we ask the user if it needs to be read as an option before the documentation pass through the file. It's more convenient for the user to answer all such questions at the beginning, so we build, in this macro, a token list which contains the letters 'y' or 'n', in order corresponding with the file names, with 'y' indicating that the file *does* need to be read. Therefore, this macro needs to be initialized to `\@empty`.

```
\let\styleoption=\@empty
```

**\docoption** This macro similarly records whether the file is being fully documented, or whether the `\OnlyDescription` command should be invoked.

```
\let\docoption=\@empty
```

**\crossoption** And this one records whether cross-referencing shall be enabled during the processing of the document.

```
\let\crossoption=\@empty
```

**\ifyes** This is a pseudo-if, which takes one argument, a command into which a user response has been read by `\typein`. It behaves like `\iftrue` only if the response commences with the letter 'y', in either lower- or upper-case.

Our first action, therefore, is to force all the characters of the response to be lower-case.

```
\def\ifyes#1{%
  \edef\next{\def\noexpand #1{#1}}%
  \lowercase\expandafter{\next}%
```

Now we strip the response down to just its first character, so that we can make the comparison correctly.

```
\edef #1{\expandafter\firstch@r #1\p@ramend}%
```

We finish up by making the comparison.

```
\if #1y}
```

`\firstch@r` These macros yield, respectively, the first (or only) character, and the remaining characters, in the token string provided as argument. This list has to be terminated by the token `\p@ramend`.

```
\def\firstch@r#1#2\p@ramend{#1}
\def\otherch@rs#1#2\p@ramend{#2}
```

### 2.1.1 Recognizing parts of a file specification

As mentioned above, it is necessary to be able to recognize files whose file type is given as `.sty`; `autodoc` adopts a very simplistic approach to this, attempting to recognize the characters which follow the first period in the `\docname`, so cannot handle, for example, complicated `VAX/VMS` directory specifications. At the expense of making this style file operating system specific, it would be possible to extend this code to be able to strip off parts of the file specification which precede the file name itself, but it is recommended instead that all the files being documented should either be in the current directory or in the directory where `TEX` expects to find standard inputs (`TeXinputs:`), so that an explicit directory or path does not need to be given.

`\filetype` This macro yields (in `\ext`) the characters (if any) which follow the *first* period in its argument; the latter has to be terminated by `\p@ramend`.

```
\def\filetype#1.#2\p@ramend{\def\ext{#2}}
```

`\filen@me` This macro yields whatever *precedes* the first period in its argument, which again is terminated by the token `\p@ramend`.

```
\def\filen@me #1.#2\p@ramend{#1}
```

`\@ifextsty` This macro firstly tests to see if its argument includes the characters `.sty` at its end. If so, it interacts with the user to determine whether this is a *style option* which needs to be read (to establish its macros) before it may be documented. If this is the case, the macro sets `\@tempwatrue`.

We start by assuming that the given argument doesn't have any file extension. The macro `\ext` will be set to the string of characters which follow the first period in `\docname`, and if there is no such period, to `'tex'`.

```
\def\@ifextsty#1%
\expandafter\filetype #1.tex\p@ramend
```

If there *is* an explicit file extension, then `\ext` will *not* be `'tex'`. This next bit gets any such actual extension into `\ext`.

```
\ifx\ext\ext@is@tex
\else
\expandafter\filetype #1\p@ramend
\fi
```

We can now set `\@tempwatrue` if those last three characters are `'sty'`.

```
\ifx\ext\ext@is@sty \@tempwatrue \fi
```

If the file type *is* `.sty`, we ask the user whether the file should be read as a style option. This has to be done by a separate macro, because the pseudo-if `\if@yes` will not be correctly matched with its `\fi` if the file type *isn't* `.sty`.

```
\if@tempwa
\testif@ption
\fi
```

```
}
```

`\testif@ption` This macro is invoked if `\fulldocname` has been found to end in `.sty`. It asks the user whether it needs to be read as a style option.

```
\def\testif@ption{%
  \typein[\is@option]{Does the description of \fulldocname\space
                    use its macros? }
```

Now we use `\if@yes` to determine whether `\is@option` was an affirmative response. If a negative response was given, we cancel `\@tempwatrue`.

```
\if@yes \is@option \else
  \@tempwafalse
\fi}
```

`\ext@is@tex` These macros expand to the strings 'tex' and 'sty', respectively, and are used in recognizing these strings if they form part of the current argument of `\@ifextsty`.

```
\ext@is@sty
\def\ext@is@tex{tex}
\def\ext@is@sty{sty}
```

`\SaveAnswer` This macro takes two arguments; the first is whatever the user has typed in as a response to a `\typein` command, whilst the second gives the name of a command which is to have the letter 'y' or 'n' appended, depending upon whether the first argument is an affirmative answer.

```
\def\SaveAnswer#1#2{%
```

We use `\if@yes` to extract from the response its first character, converted to lowercase, and to compare this with 'y'.

```
\if@yes #1
```

If this response is affirmative, we append a 'y' to the second argument.

```
\edef #2{#2y}%
```

Alternatively, we append an 'n'.

```
\else
  \edef #2{#2n}%
\fi}
```

`\Check@ption` Again, this pseudo-if test has to be placed inside a separate macro if  $\TeX$  is not to become confused when skipping conditional text.

```
\def\Check@ption{%
  \typein[\is@option]{Can reading of \fulldocname\space be deferred
                    until it is processed? }
  \if@yes \is@option
  \else
    \input \fulldocname\relax
  \fi}
```

## 2.2 Scanning the file names in `\docnames`

`\thisdoc` Now we are ready to cycle through all the file names in `\docnames`, during which  
`\thedoc` we shall establish whether the file name ends in `.sty`, and if so, whether the user wants it to be read as a style option. However, we *know* that the files `doc.sty` and `autodoc.sty` should *not* be read in as options, so we establish here two macros which expand to those names, so that they may be used in `\ifx` tests.

```
\def\thisdoc{autodoc.sty}
\def\thedoc{doc.sty}
```

`\fulldocname` At last we start to cycle through the filenames and get the user's responses.

```
\@for\fulldocname:=\docnames\do{%
```



We assume initially that the file *shouldn't* be read as an option...

```
\@tempswafalse
```

Now we don't want to read **this** file again (otherwise we'll recurse and annoy the user by repeatedly asking for the name of the option to be documented)! Therefore we avoid that here...

```
\ifx\fulldocname\thisdoc
\else
```

Nor do we want to read the doc style option yet, because we're going to read it in at the end of this file.

```
\ifx\fulldocname\thedoc
\else
```

Otherwise, we use \@ifextsty to establish whether it *is* a file with suffix .sty, and, if so, whether it should be read as an option.

```
\@ifextsty\fulldocname
\fi
\fi
```

At this point, we know whether the file should be read in later as a style option: we put the appropriate character into \style@option.

```
\if@tempswa
\Check@option
\else
\def\is@option{n}
\fi
\SaveAnswer{\is@option}{\style@option}
```

Now we ask whether the file is to be fully documented.

```
\typein[\is@full]{Should \fulldocname\space
be fully documented? }%
\SaveAnswer{\is@full}{\doc@option}
```

For our final question regarding each file being documented, we ask if full cross-referencing is required. Processing is *much* faster without this.

```
\typein[\is@cross]{Should \fulldocname\space
be cross referenced? }%
\SaveAnswer{\is@cross}{\cross@option}
```

And that completes the preliminary processing of the names in \docnames.

```
}
```

**\options** The user is asked one final question before the doc option is read in: whether additional options are required.

```
\typein[\options]{Give the names of any further
options (WITHOUT .sty): }%
```

If a non-null reply is given, each such option is read in...

```
\if \options \endlinechar
\else
\@for\fulldocname:=\options\do{\input\fulldocname.sty\relax}%
```

Otherwise, there's nothing more to be done.

```
\fi
```

### 3 Processing the specified files

`\processdocs` This is the macro which, when invoked as the body of the document, will cause all the files to be input and documented.

We commence by cycling through each of the files to be processed. Any preliminary material (for instance, a description of the files being described) will use a setting of zero for TeX's `\count1` register. Before we do this, however, we must remember the `.toc` file for the complete document, so that entries may be directed to it at the end of processing each document.

```
\def\processdocs{\count1=0
\let\SavedTOC=\tf@toc
\@for\fulldocname:=\docnames\do{%
```

`\docname` We then parse the filename to extract just the name into `\docname`, and generate an  
`\Docname` uppercase version of it for use in page numbering.

```
\edef\docname{\expandafter \file@me \fulldocname .tex\p@ramend}%
\edef\next{\def\noexpand\Docname{\docname}}%
\uppercase\expandafter{\next}%
```

At this point, we start a new group for each file being documented, in order that any changes to macros defined in doc remain local to each processed file. We also ensure that any writes to a `.toc` file are discarded unless the individual document has opened one.

```
\begingroup
\let\tf@toc=\relax
```

Now we look at the first character of `\style@ption` to determine whether the file shall be read as a style option...

```
\expandafter\expandafter\expandafter \if
\expandafter\firstch@r \style@ption\p@ramend y
```

If it is to be read, we suspend doc's ignorance of `%` whilst it is read, and treat `@` as a letter, so that we are reproducing the conditions that pertain when style options are ordinarily read.

```
\MakePercentComment\makeatletter
\input\fulldocname\relax
\MakePercentIgnore\makeatother
\fi
```

Throw away that first character of `\style@ption`, which will therefore now commence with the user's response relating to the *next* file to be processed.

```
\xdef\style@ption{\expandafter \otherch@rs \style@ption\p@ramend}%
```

#### 3.1 Processing the file itself

We start by changing the `\jobname` appropriately, in order that the correct `.toc`, `.ind`, etc., files can be read. We also ensure that the next printed page will be *recto*, since it's nicer to start a new document on a right-hand page.

```
\edef\jobname{\docname}\clearpage
\ifodd\count0\else
\vspace*{\fill}%
\centerline{\small This page is left intentionally blank}%
\vspace*{\fill}\clearpage
\fi
```

Pages are numbered individually within each separate document, so reset the counter and redefine `\@oddfn` to include the document name in the page numbers. We also

use T<sub>E</sub>X's `\count1` register to give us *different* page numbers, to facilitate the use of DVI processing programs. Note that we do *not* do this through `\thepage`, because that would also affect the table of contents and index entries.

```
\setcounter{page}{1}\global\advance\count1by1\relax
\def\@oddfoot{\hfil\Docname--\thepage\hfil}%
\let\@evenfoot=\@oddfoot
```

At this point, we perform most of the actions that would be undertaken by `\include`; the latter cannot be used for this task, because it is *only* capable of reading files with type `.tex`.

A command is put into the main document's auxiliary file to read the `.aux` file for the document being processed; when the main `.aux` file is being read, we prevent entries being written to the main document's table of contents, etc, which actually relate to the tables of the included file(s).

```
\if@filesw
\immediate\write\@mainaux{\string\let\string\tf@toc
                          =\string\relax}%
\immediate\write\@mainaux{\string\@input{\docname.aux}}%
\immediate\write\@mainaux{\string\let\string\tf@toc=
                          \string\SavedTOC}%
```

We arrange for output to an auxiliary file to go to the appropriate file:

```
\immediate\openout\@partaux \docname.aux
\let\@auxout=\@partaux
```

We need to start that off tidily...

```
\immediate\write\@partaux{\relax}\fi
```

Now we examine `\cross@option` to decide whether cross-referencing shall be enabled...

```
\expandafter\expandafter\expandafter
\if \expandafter \firstch@r \cross@option \p@ramend y
\EnableCrossrefs
\else
\DisableCrossrefs
\fi
\xdef\cross@option{\expandafter\otherch@rs\cross@option\p@ramend}%
```

And `\doc@option` determines whether the entire document is to be processed, or whether `\OnlyDescription` is required.

```
\expandafter\expandafter\expandafter
\if \expandafter \firstch@r \doc@option \p@ramend y
\else
\OnlyDescription
\fi
\xdef\doc@option{\expandafter\otherch@rs\doc@option\p@ramend}%
```

We activate the recording of modification records and of an index. Now we are in a position to read the next file being documented. Afterwards, we revert to single-column setting again (in case an index has been printed).

```
\RecordChanges
\MakeIndex
\input{\fulldocname}\clearpage \onecolumn
```

By reverting to the original auxiliary file and reading back that just written, we complete the toc entries, etc. When performing this read, we do it under the same conditions as for an `\end{document}`, which redefines a number of commands; however, we're already inside a group that is about to be ended, so there's no need to start another.

```
\let\@auxout=\@mainaux
```

```

\if@filesw \immediate\closeout\@partaux
  \def\global\@namedef##1##2{}\def\newlabel{\@testdef r}%
  \def\bibcite{\@testdef b}\@tempwafalse
  \makeatletter\input \docname.aux
\fi

```

After this, we'd better close any .toc files, etc. We close the group first, because these changes should apply "globally" to each file being documented.

```

\endgroup
\@for\ext:=\extensions\do{\@closetoc{\ext}}%

```

We also close any glossary and index files that may be open, because T<sub>E</sub>X can only manage a limited number of simultaneously open files.

```

\if@filesw \immediate\closeout\@glossaryfile
  \immediate\closeout\@indexfile \fi

```

Now L<sup>A</sup>T<sub>E</sub>X's \makeindex and doc's \RecordChanges commands both allocate the files' "handles" through the \newwrite command, and it would be wasteful to keep on allocating new files through this mechanism. So we redefine \makeindex and \RecordChanges to open new files, appropriate to the file being processed, but reusing the same "handle" (providing we are actually generating auxiliary files).

```

\let\makeindex=\LaterMakeIndex
\let\RecordChanges=\LaterRecordChanges

```

We are now able to write (to the driver file's .aux file, and thence to its .toc file) an entry for the complete document's table of contents. We use L<sup>A</sup>T<sub>E</sub>X's \addtocontents for this, but being impatient sorts, we need the \write to take place \immediately (actually, we're not really impatient; without this the contents for the last document processed would be stuck in a "whatsit", and never written to the file, because \output will not be called again). Therefore, we make local redeclarations for \write.

```

{\let\Write=\write \def\write{\immediate\Write}%
\addtocontents{toc}{\protect \contentsline{section}%
{\protect\numberline{\number\count1}\savedtitle}{\Docname--1}}}%

```

Finally, before reading any further files, we'd better reset the section counters, and cancel the effect of any \appendix command. We zero the footnote counter as well, in case a subsequent \title has a \thanks on it.

```

\setcounter{section}{0}%
\setcounter{subsection}{0}%
\def\thesection{\arabic{section}}%
\setcounter{footnote}{0}%

```

And that's all there is to it!!!! As our parting shot, we ensure that any \@writefile commands that may be read from the .aux files will be directed to the correct toc file.

```

\let\tf@toc=\SavedTOC
}% end of the \do
}

```

### 3.2 Handling toc files, etc

**\@closetoc** We need to be able to close toc files, etc., for each file being documented. Files are only closed if they exist, and have been used for the current document!

```

\def\@closetoc#1{\ifundefined{tf@#1}{\expandafter
\ifx \csname tf@#1\endcsname \relax \else
\expandafter \immediate \closeout \csname tf@#1\endcsname
\typeout{Closing \docname.#1 file}\fi

```

Since it is not possible to cancel the effect of the `\newwrite` which allocated the `toc` file, we merely make L<sup>A</sup>T<sub>E</sub>X's internal reference to the file be `\relax`.

```
\global\expandafter\let\csname tf@#1\endcsname=\relax}%
}
```

`\extensions` Here is the list of possible file extensions used for table of contents files, etc.

```
\def\extensions{toc,lof,lot}
```

`\@writefile` Because `toc` files, etc., are allocated by `\newwrite`, which defines the “name” globally, we need a modified `\@writefile` that suppresses output if the “name” has been “undefined” (by letting it be `\relax`). This can happen

- if no file has *ever* been opened; or
- if the file has been closed, and not re-opened.

```
\def\@writefile#1#2{\ifundefined{tf@#1}{}\expandafter
\ifx\csname tf@#1\endcsname \relax \else
\expandafter\immediate\write\csname tf@#1\endcsname{#2}\fi}}
```

`\savedtitle` When documenting a number of files, it's pleasant to be able to put a table of contents before all the documented files, listing the titles of the files documented. Therefore, we need to save each document's title until after it has been processed. It is saved in this macro, which is here given a default definition.

```
\def\savedtitle{\hbox{}}
```

`\@maketitle` We give it a value when `\@maketitle` is called<sup>3</sup>, so we save here the original definition of that macro, and define a new one which makes the necessary save operation inside a group: when doing that, we want to discard any `\thanks` that might be in the user's `\title` command.

```
\let\orig@maketitle=\@maketitle
\def\@maketitle{\def\protect{\noexpand\protect\noexpand}%
\let\thanks=\@gobble
\xdef\savedtitle{\@title}}%
\orig@maketitle}
```

`\LaterMakeIndex` Here are the redefinitions for `\makeindex` and `\RecordChanges`. They are similar to the ordinary `\makeindex`, but re-use the same file “handle” (providing we are actually generating auxiliary files). Note that we have to make the appropriate definitions for `\index` and `glossary`

```
\def\LaterMakeIndex{\if@filesw
\immediate\openout\@indexfile=\docname.idx\relax
\def\index{\@bsphack\begingroup
\def\protect####1{\string####1\space}\@sanitize
\@wrindex\@indexfile}\typeout
{Writing index file \docname.idx}%
\fi}%
\def\LaterRecordChanges{\if@filesw
\immediate\openout\@glossaryfile=\docname.glo\relax
\def\glossary{\@bsphack\begingroup\@sanitize
\@wrindex\@glossaryfile}\typeout
{Writing glossary file \docname.glo}%
\fi}%
```

<sup>3</sup> It would have been cleaner to have redefined `\maketitle` to have done the work, but that gets redefined in `doc`, which we haven't yet read!

#### 4 Reading in the doc Style Option

Finally, as our parting shot, we read in the doc style option, which will set up everything for creating the documentation. Before we do so, however, we “remember” the `\makeindex` command (because ordinarily that may be issued only in the preamble), and define `\DocstyleParms` to do nothing; this will prevent the doc style from invoking `\makeindex` unnecessarily for the root file.

```
\let\MakeIndex=\makeindex
\def\DocstyleParms{\relax}
```

We now read in `doc.sty`; that will leave the `%` character ignorable, so we’ll set it back to it’s usual state before tidying up this file, which ends, (you’ll have to take my word for it!) with a call of `\MakePercentIgnore` so that documentation can proceed in the usual way.

```
\input{doc.sty}
\MakePercentComment
```

#### References

[1] F. MITTELBACH The doc-Option. (see page 245 of this issue of TUGboat.).

#### Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

	<b>Symbols</b>	<code>\ext@is@tex</code> . . . . . <u>278</u>	<code>\MakePercentIgnore</code> . 280
	<code>\@closetoc</code> . . . . . <u>282</u>	<code>\extensions</code> . . . . . <u>283</u>	
	<code>\@ifextsty</code> . . . . . <u>277</u>		<b>O</b>
	<code>\@maketitle</code> . . . . . <u>283</u>	<b>F</b>	<code>\OnlyDescription</code> .. 281
	<code>\@writefile</code> . . . . . <u>283</u>	<code>\filedate</code> . . . . . 276	<code>\options</code> . . . . . <u>279</u>
	<b>A</b>	<code>\filen@me</code> . . . . . <u>277</u>	<code>\orig@maketitle</code> ... <u>283</u>
	<code>\addtocontents</code> .... 282	<code>\filetype</code> . . . . . <u>277</u>	<code>\otherch@rs</code> . . . . . <u>277</u>
	<b>C</b>	<code>\fileversion</code> . . . . . 276	<b>P</b>
	<code>\Check@ption</code> . . . . . <u>278</u>	<code>\firstch@r</code> . . . . . <u>277</u>	<code>\processdocs</code> .. 274, <u>280</u>
	<code>\cross@ption</code> . . . . . <u>276</u>	<code>\fulldocname</code> . . . . . <u>278</u>	<b>R</b>
	<b>D</b>	<b>G</b>	<code>\RecordChanges</code> 281, 282
	<code>\DisableCrossrefs</code> . 281	<code>\glossary</code> . . . . . 283	<b>S</b>
	<code>\doc@ption</code> . . . . . <u>276</u>	<b>I</b>	<code>\SaveAnswer</code> . . . . . <u>278</u>
	<code>\Docname</code> . . . . . <u>280</u>	<code>\if@yes</code> . . . . . <u>276</u>	<code>\savedtitle</code> . . . . . <u>283</u>
	<code>\docname</code> . . . . . <u>280</u>	<b>L</b>	<code>\style@ption</code> . . . . . <u>276</u>
	<code>\docnames</code> . . . . . <u>276</u>	<code>\LaterMakeIndex</code> ... <u>283</u>	<b>T</b>
	<b>E</b>	<code>\LaterRecordChanges</code> <u>283</u>	<code>\testif@ption</code> . . . . . <u>278</u>
	<code>\EnableCrossrefs</code> .. 281	<b>M</b>	<code>\thedoc</code> . . . . . <u>278</u>
	<code>\ext@is@sty</code> . . . . . <u>278</u>	<code>\MakePercentComment</code> ..... 280, 284	<code>\thisdoc</code> . . . . . <u>278</u>

◊ B Hamilton Kelly  
Royal Military College of Science  
Shrivenham  
SWINDON  
SN6 8LA  
United Kingdom  
Janet: `tex@uk.ac.cranfield.rmcs`

## Calendar

**1989**

**Harvard University,  
Cambridge, Massachusetts**

Jun 5-9 Advanced T<sub>E</sub>X/Macro Writing  
Jun 6-9 Intermediate T<sub>E</sub>X

---

**University of Maryland,  
College Park, Maryland**

Jun 5-9 Intensive Beginning/Intermed. T<sub>E</sub>X  
Jun 12-16 Intensive L<sup>A</sup>T<sub>E</sub>X

---

Jun 12 Nordic T<sub>E</sub>X meeting  
Royal Institute of Technology,  
Stockholm (See page 287.)

---

**University of Illinois, Chicago, Illinois**

Jun 12-16 Intensive Beginning/Intermed. T<sub>E</sub>X  
Jun 13-16 Beginning T<sub>E</sub>X

---

**Providence College,  
Providence, Rhode Island**

Jun 12-16 Intensive Beginning/Intermed. T<sub>E</sub>X  
Jun 13-16 Beginning T<sub>E</sub>X

---

**University of Michigan,  
Ann Arbor, Michigan**

Jun 19-23 Intensive Beginning/Intermed. T<sub>E</sub>X  
Jun 20-23 Beginning T<sub>E</sub>X  
Jul 6-7 Macro Writing  
Jul 6-7 Output Routines  
Jul 10-14 Advanced T<sub>E</sub>X/Macro Writing  
Jul 11-14 Intermediate T<sub>E</sub>X

---

**McGill University, Montréal, Québec**

Jun 19-23 Advanced T<sub>E</sub>X/Macro Writing  
Jun 20-23 Intermediate T<sub>E</sub>X  
Jun 26-30 Intensive L<sup>A</sup>T<sub>E</sub>X  
Jul 3-7 Intensive Beginning/Intermed. T<sub>E</sub>X  
Jul 4-7 Beginning T<sub>E</sub>X

---

**Northeastern University,  
Boston, Massachusetts**

Jun 19-23 Intensive Beginning/Intermed. T<sub>E</sub>X  
Jun 20-23 Beginning T<sub>E</sub>X  
Jun 29-30 L<sup>A</sup>T<sub>E</sub>X, T<sub>E</sub>X courses, NTG-“happening”  
Utrecht. For information, contact  
C. G. van der Laan (Bitnet:  
cgl@HGRRUG5).

---

Jul 5 UKTUG: “Fonts, Design and Use”  
Aston University, Birmingham, U.K.  
Jul 10-14 Intensive Beginning/Intermed. T<sub>E</sub>X  
Exeter University, England  
Jul 30- Aug 4 ACM SIGGRAPH '89, Boston,  
Massachusetts. Contact: Chris Herot  
or Branko Gerovac, (312) 644-6610.

---

**Rutgers University, Busch Campus,  
Piscataway, New Jersey**

Jul 31- Aug 4 Intensive Beginning/Intermediate  
T<sub>E</sub>X  
Aug 1-4 Beginning T<sub>E</sub>X  
Aug 7-11 Advanced T<sub>E</sub>X/Macro Writing  
Aug 8-11 Intermediate T<sub>E</sub>X  
Aug 14-18 Intensive L<sup>A</sup>T<sub>E</sub>X

---

Aug 14-18 Intensive L<sup>A</sup>T<sub>E</sub>X, University of  
Michigan, Ann Arbor, Michigan

---

**T<sub>E</sub>X Users Group 1989 Conference  
— Tenth Anniversary —**

**Stanford University, Stanford, California**  
Aug 14-18 Intensive Beginning/Intermed. T<sub>E</sub>X  
Aug 15-18 METAFONT  
Aug 16-18 PostScript  
Aug 19 Introduction to SGML  
Aug 20-23 **TUG Annual Meeting**  
Aug 24-25 Graphic Design of  
Technical Documents  
Aug 24-25 Macro Writing  
Aug 24-25 Output Routines

---

**Vanderbilt University, Nashville, Tennessee**

- Aug 14–18 Intensive Beginning/Intermed.  $\TeX$   
 Aug 15–18 Beginning  $\TeX$   
 Aug 21–25 Advanced  $\TeX$ /Macro Writing  
 Aug 22–25 Intermediate  $\TeX$
- 

**University of Illinois, Chicago, Illinois**

- Aug 14–18 Intensive  $\text{L}\text{A}\text{T}\text{E}\text{X}$   
 Aug 28–31  $\TeX$  Wizard Course  
 Sep 11–14 METAFONT  
 Sep 12–15 Intermediate  $\TeX$
- 

- Sep 7–8 Macro Writing, University of Maryland, College Park, Maryland  
 Sep 11 **TUGboat Volume 10, No. 3:** Deadline for receipt of manuscripts.  
 Sep 11–13  $\TeX$ 89: 4th Annual Meeting of European  $\TeX$  Users, University of Karlsruhe, FRG. For information, contact Rainer Rupprecht (Bitnet: RZ32@DKAUNI48. See also TUGboat 10#1, page 118.)
- 

**Exeter University, England**

- Sep 18–22 Advanced  $\TeX$ /Macro Writing  
 Sep 26–29 Beginning METAFONT
- 

- Oct 3–6 Protex V Conference: 5th International Conference on Computer-Aided Text Processing and its Applications. Boston, Massachusetts. For information, contact Protex Conference, INCA, P. O. Box 2, Duń Laoghaire, Ireland; +353-1-613749.  
 Oct 11–13 DANTE—Deutschsprachige Anwendervereinigung  $\TeX$ . Eichstätt, FRG. For information, contact Joachim Lammarsch (Bitnet: RZ92@DHDURZ1. See also page 287.)

- Oct 12–13 RIDT'89—Raster Imaging and Digital Typography. Ecole Polytechnique Fédérale, Lausanne, Switzerland. For information, contact Prof. R.D. Hersch, Lausanne, Switzerland (hersch@elde.epfl.ch or (4121) 693 43 57); or Debra Adams (adams.pa@Xerox.com or (415) 494-4022). (See announcement, TUGboat 9#3, page 316.)  
 Nov 23 4<sup>e</sup> NTG-Byeenkomst. Tilbing, The Netherlands. For information, contact H. Mulders of C. G. van der Laan (Bitnet: cgl@HGRRUG5).
- 

**1990**

- Jan 15 **TUGboat Volume 11, No. 1:** Deadline for receipt of manuscripts (tentative).  $\TeX$ 90, Cork, Ireland.  
 Sep 18–20 EP'90 National Institute of Standards and Technology, Gaithersburg, Maryland. For information, contact Richard Furuta (furuta@brillig.umd.edu).

For additional information on the events listed above, contact the TUG office (401-751-7760) unless otherwise noted.



---

**NORDIC T<sub>E</sub>X Meeting**

Stockholm, June 12th, 1989

**Location**

Lindstedtsvägen 25  
 Room D1  
 Royal Institute of Technology  
 Stockholm

**Program**

T<sub>E</sub>X today and in the future. Niels Mortensen  
 Back to international standards. Malcolm Clark  
 Nordic national letters. Staffan Romberger  
 Standard proposal (recommendations for TUG 89)  
 T<sub>E</sub>X and graphics. Bjørn Larsen  
 Integration of Autocad in L<sup>A</sup>T<sub>E</sub>X. Anders Eriksson  
 Graphics in T<sub>E</sub>Xtures. Miro Sedlacek  
 T<sub>E</sub>X on workstations (demo of the Publisher).  
 Phototypesetting and high quality output.

Peter Busk Laursen

T<sub>E</sub>X and PostScript — experiences. Jan M Rynning  
 Panel discussion.

Dinner at the KTH “Värdshus”

**For information, contact**

Roswitha Graham  
 tel: 46 8 790 6525  
 fax: 46 8 25 10  
 tlx: 103 89 KTHB S  
 email: roswitha@admin.kth.se

---

**L<sup>A</sup>T<sub>E</sub>X course book in Dutch**

As one of the results of the Dutch activities with respect to T<sub>E</sub>X a (Dutch, but read on!) course book on L<sup>A</sup>T<sub>E</sub>X has emerged. It is called

*Publiceren with L<sup>A</sup>T<sub>E</sub>X*

by R. de Bruin, C.G. van der Laan,  
 J.R. Luyten, H.F. Vogt

CWI-syllabus 19. Centrum voor Wiskunde  
 en Informatica, Postbus 4079 1009 AB Am-  
 sterdam, Holland. (196pp) (Price: fl. 28.50,  
 posting costs not included)

Although it is written in Dutch, the exercises are primarily directed to report preparation by the aid of computers, especially L<sup>A</sup>T<sub>E</sub>X. So, the exercises and especially the answers are useful for a broader audience.

---

**DANTE — Deutschsprachige  
Anwendervereinigung T<sub>E</sub>X**

Joachim Lammarsch

Last year, when I was in Exeter at the European T<sub>E</sub>X meeting, was the first time that the idea of a German T<sub>E</sub>X society came into my mind. I saw that the organisation and representation of the European T<sub>E</sub>X users was not very good. And the first step toward making it better is in my opinion to improve the organisation of the national groups.

So because of that, in Freiburg at the German T<sub>E</sub>X meeting, I made the proposal in my capacity as the coordinator of the German group to found a German T<sub>E</sub>X society, and asked for comments about that idea. The response has been so positive that I have gone on to prepare the foundation.

On 14th of April 1989 in Heidelberg, it has come to fruition. DANTE, Deutschsprachige Anwendervereinigung T<sub>E</sub>X, has been founded.\* 17 persons were present to found the society, and after a discussion about the statutes the board was elected. For chairman of the society the persons present elected me. For vice chairwoman Mrs. Gabriele Kruljac, Max-Planck-Institut Stuttgart, for treasurer Mr. Friedhelm Sowa, Research Center of the University Düsseldorf, and for secretary Mrs. Luzia Dietsche, Research Center of the University Heidelberg, have been chosen.

The principal aim of the society is to encourage advice and cooperation among German language T<sub>E</sub>X users. But this is not the only intention. The user group will examine proposals of members for new T<sub>E</sub>X software, if there are some. It will above all cooperate with other related national and international T<sub>E</sub>X groups. Besides, DANTE shall represent the interests of the German language T<sub>E</sub>X users to the TUG more than has happened up till now. Perhaps that will be done in team-work with other European T<sub>E</sub>X groups.

Another activity will be the organisation of training and education. The first training course will perhaps be held at the next German T<sub>E</sub>X meeting. Last, but not least, a newspaper will be edited and published.

Institutions as well as individuals can become members. Membership is possible for universities, publishers, software houses, computer companies,

---

\* I agree, the name is not from a German. But the French users group has taken the German GUTenberg, so we needed another well-known name.

public authorities, private persons, students, e.a. to name but a few. The dues for the various groups are graduated.

The first general meeting will take place together with the German T<sub>E</sub>X meeting on 11th-13th October 1989 in Eichstätt. The first day of the meeting will be reserved for the members of DANTE; the other two days will be the same as normally happens at meetings.

For more information about DANTE please contact:

DANTE - Deutschsprachige Anwender-  
vereinigung T<sub>E</sub>X  
Research Center of the University Heidelberg  
Im Neuenheimer Feld 293  
6900 Heidelberg 1  
West Germany  
Bitnet: DANTE@DHDURZ1

or

Joachim Lammarsch  
Research Center of the University Heidelberg  
Im Neuenheimer Feld 293  
6900 Heidelberg 1  
West Germany  
Bitnet: RZ92@DHDURZ1

---

### Notes on first meeting of UK T<sub>E</sub>X Users' Group, 15 March 1989

David Osborne

The London School of Economics was the venue on 15th March for the first meeting of a UK T<sub>E</sub>X Users' group (numbering November's inaugural Nottingham meeting as the 'zeroth'<sup>1</sup>). Unlike the earlier meeting, the LSE meeting took a theme for all the talks: "Graphics and T<sub>E</sub>X". Because more time had been available to plan and publicise the event, it was good to see a broader representation of T<sub>E</sub>X users instead of the bias towards those from academe and a rough head-count gave between 40 and 50 attendees.

---

<sup>1</sup> see TUGboat 10,1 for a report on the Nottingham meeting.

### "Business meeting"

(Malcolm Clark, Imperial College)

As the gathering was intended to be the first official meeting of a national UK user group, Malcolm Clark began with what he called a "business meeting" to discuss matters relating to the organisation of a user group, such as a constitution and the formation of an organising committee. Malcolm had prepared a draft constitution for the group, copies of which he distributed. This covered not just administrative matters, such as arrangements for annual meetings and the number of members at these to form a quorum, but also raised some questions which he asked those present to consider and decide at the next meeting. These covered important issues such as the relationship of a UK group to other European T<sub>E</sub>X user groups and to TUG itself, together with the financial (and political) implications of membership fees. Clearly, these will require a good deal of discussion to sort out but Malcolm argued for co-operation with other groups, where possible. Communication of information between users in the group and between this and other groups is obviously an important point and some suggestions were made regarding a newsletter. Malcolm offered to make the T<sub>E</sub>Xline mailing list available, though T<sub>E</sub>Xline will continue as an independent publication. An *ad hoc* election of a committee to organise the group's future activities was then conducted.

Finally, to lead in to the theme of the day's meeting, Malcolm distributed (T<sub>E</sub>Xed) copies of a recent article by David Rogers, which had first appeared in the T<sub>E</sub>Xhax digest,<sup>2</sup> later printed in TUGboat, "Computer Graphics and T<sub>E</sub>X — A Challenge".

### Technical Program

Editor's note: Owing to limited space in this issue, the details of the technical program have been omitted, and only the speakers and their topics are listed.

- **Picture Languages** (*Sebastian Rahtz, Southampton University*)
- **PostScript** (*David Morgan, Imperial College*)
- **PostScript in T<sub>E</sub>X** (*David Brightly, Daresbury Laboratory*)
- **Chemical diagrams** (*Tony Norris, Polytechnic of the South Bank, London*)
- **Typesetting pictures in T<sub>E</sub>X** (*Adrian Clark, University of Essex*)

---

<sup>2</sup> T<sub>E</sub>Xhax V89 #8, 19 Jan 1989.

Following a question and answer session, the meeting closed with thanks being expressed to Carol Hewlett of LSE who, with Malcolm Clark, had arranged the meeting. The date of the next meeting wasn't announced, but has subsequently appeared in the bytes of UKTEX. It will take place on Wednesday, July 5th at Aston University, Birmingham, the theme being "Fonts—Design and Use". The following meeting is provisionally arranged to be in October 1989 in London.

---

## Production Notes

Barbara Beeton

### Input and input processing

Electronic input for articles in this issue was received by mail and on floppy disk. Camera copy was accepted for one article and for several figures (see the "output" section).

Authors who had written articles previously for TUGboat typically submitted files that were fully tagged and ready for processing with the TUGboat macros—`tugbot.sty` for PLAIN-based files and `ltugbot.sty` for those using L<sup>A</sup>T<sub>E</sub>X.

Much-improved versions of the TUGboat macros were available for this issue, thanks to the efforts of Ron Whitney. Most L<sup>A</sup>T<sub>E</sub>X-based articles can now be run in a stream controlled from a driver file (this has always been true for PLAIN-based items). The exceptions are articles formatted with the `doc-option` of L<sup>A</sup>T<sub>E</sub>X created by Frank Mittelbach (identified below).

About a third of the articles, and almost half the pages in this issue are L<sup>A</sup>T<sub>E</sub>X. For convenience in processing, PLAIN or L<sup>A</sup>T<sub>E</sub>X articles were grouped whenever possible. Articles in which no, or limited, T<sub>E</sub>X coding was present were tagged according to the conventions of `tugbot.sty` or `ltugbot.sty` as convenient. Articles tagged according to the author's own schemes were modified sufficiently to permit them to be merged with the rest of the stream. Especial care was taken to try to identify macro definitions that conflicted with ones already defined for TUGboat, and `\begingroup ... \endgroup` was wrapped around any suspect article as a routine precaution. Even so, several "time bombs" made themselves known; a particularly

insidious one is the subject of a note at the end of the article by David Salomon (page 207).

Test runs of articles were made separately and in groups to determine the arrangement and page numbers (to satisfy any possible cross references). A file containing all starting page numbers, needed in any case for the table of contents, was compiled before the final run. Final processing was done in one run of T<sub>E</sub>X and four of L<sup>A</sup>T<sub>E</sub>X, using the page number file for reference.

The following articles were prepared using L<sup>A</sup>T<sub>E</sub>X; the starred items required the `doc-option`.

- Don Hosek, *Guidelines for creating portable METAFONTcode*, page 173.
- Tom Reid and Don Hosek, *Report from the DVI driver standards committee*, page 188.
- Malcolm Clark, *T<sub>E</sub>Xline: A newsletter of the T<sub>E</sub>X community*, page 193.
- Don Hosek, *Announcing (belatedly) T<sub>E</sub>XMA<sub>G</sub>*, page 192.
- Peter Abbott, *UKT<sub>E</sub>X and the Aston archive*, page 194.
- Ted Nieland, *The DECUS T<sub>E</sub>X collection*, page 195.
- \* Frank Mittelbach and Rainer Schöpf, *A new font selection scheme...*, page 222.
- Dezső Nagy, *A bar chart in L<sup>A</sup>T<sub>E</sub>X*, page 239.
- Hubert Partl, *Producing on-line information files with L<sup>A</sup>T<sub>E</sub>X*, page 241.
- \* Frank Mittelbach, *The doc-option*, page 245.
- \* B. Hamilton Kelly, *The autodoc-option*, page 274

### Output

The bulk of this issue was prepared at the American Mathematical Society on a VAX 6320 (VMS) and output on an APS- $\mu$ 5 using resident CM fonts and additional downloadable fonts for special purposes. The items listed below were received as camera copy; they were prepared on the devices indicated. The output devices used to prepare the advertisements were not usually identified; anyone interested in determining the device used for a particular ad should inquire of the advertiser. The appearance of the printed pages can be taken as representative of output from the devices which produced them.

- Unidentified:
  - all advertisements.
- Apple LaserWriter (300 dpi):
  - David Salomon, *DDA methods in T<sub>E</sub>X*, page 207, figures only.
- Canon CX (300 dpi): Georgia Tobin, *Another dingbat idea*, page 166.

**Request for Information**

The TeX Users Group maintains a database and publishes a membership list containing information about the equipment on which TeX is (or will be) installed and about the applications for which TeX is used. This list is updated periodically and distributed to members with TUGboat, to permit them to identify others with similar interests. Thus, it is important that the information be complete and up-to-date.

Please answer the questions below, in particular those regarding the status of TeX and the hardware on which it runs. (Operating system information is particularly important in the case of IBM mainframes and VAX.) This hardware information is used to group members in the listings by computer and output device.

If accurate information has already been provided by another TUG member at your site, indicate that member's name and the same information will be repeated automatically under your name. If your current listing is correct, you need not answer these questions again. Your cooperation is appreciated.

- *Send completed form with remittance* (checks, money orders, UNESCO coupons) to:  
TeX Users Group  
P. O. Box 594  
Providence, Rhode Island 02901, U.S.A.
- *For foreign bank transfers* direct payment to the TeX Users Group, account #002-031375, at:  
Rhode Island Hospital Trust National Bank  
One Hospital Trust Plaza  
Providence, Rhode Island 02903-2449, U.S.A.
- *General correspondence* about TUG should be addressed to:  
TeX Users Group  
P. O. Box 9506  
Providence, Rhode Island 02940-9506, U.S.A.

Name: _____
Home <input type="checkbox"/> _____
Bus. <input type="checkbox"/> Address: _____
_____
_____

QTY	ITEM	AMOUNT
	1989 TUGboat Subscription/TUG Membership (Jan.-Dec.) - <b>North America</b> New (first-time): <input type="checkbox"/> \$35.00 each Renewal: <input type="checkbox"/> \$45.00; <input type="checkbox"/> \$35.00 - reduced rate if renewed before February 1, 1989	
	1989 TUGboat Subscription/TUG Membership (Jan.-Dec.) - <b>Outside North America</b> New (first-time): <input type="checkbox"/> \$45.00 each Renewal: <input type="checkbox"/> \$50.00; <input type="checkbox"/> \$45.00 - reduced rate if renewed before February 1, 1989	
	TUGboat back volumes      1980   1981   1982   1983   1984   1985   1986   1987   1988 Circle volume(s) desired:   vol. 1   vol. 2   vol. 3   vol. 4   vol. 5   vol. 6   vol. 7   vol. 8   vol. 9 Indiv. issues \$18.00 ea.      \$18   \$50   \$35   \$35   \$35   \$50   \$50   \$50   \$50	

Air mail postage is included in the rates for all subscriptions and memberships outside North America.  
Quantity discounts available on request.

TOTAL ENCLOSED: \_\_\_\_\_  
(Prepayment in U.S. dollars required)

**Membership List Information**

Institution (if not part of address): \_\_\_\_\_

Date: \_\_\_\_\_

Title: \_\_\_\_\_

Status of TeX:  Under consideration

Phone: \_\_\_\_\_

Being installed

Network address: \_\_\_\_\_

Up and running since: \_\_\_\_\_

- Arpanet     BITnet
- CSnet       uucp
- JANET      other \_\_\_\_\_

Approximate number of users: \_\_\_\_\_

Specific applications or reason for interest in TeX:

Version of TeX:

- Pascal
- C
- other (describe)

From whom obtained: \_\_\_\_\_

My installation can offer the following software or technical support to TUG:

Hardware on which TeX is used:

Please list high-level TeX users at your site who would not mind being contacted for information; give name, address, and telephone.

Computer(s)	Operating system(s)	Output device(s)
_____	_____	_____
_____	_____	_____
_____	_____	_____

# The Joy of TEX



## A Gourmet Guide to Typesetting with the AMS-TEX macro package

M. D. SPIVAK, Ph.D.

*The Joy of TEX* is the user-friendly user's guide for AMS-TEX, an extension of TEX, Donald Knuth's revolutionary program for typesetting technical material. AMS-TEX was designed to simplify the input of mathematical material in particular, and to format the output according to any of various preset style specifications.

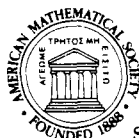
There are two primary features of the TEX system: it is a computer system for typesetting technical text, especially text containing a great deal of mathematics; and it is a system for producing beautiful text, comparable to the work of the finest printers.

Most importantly, TEX's capabilities are not available only to TEXperts. While mathematicians and experienced technical typists will find that TEX allows them to specify mathematical formulas with great

accuracy and still have control over the finished product, even novice technical typists will find the manual easy to use in helping them produce beautiful technical TEXt.

This book is designed as a user's guide to the AMS-TEX macro package and details many features of this extremely useful text processing package. Parts 1 and 2, entitled "Starters" and "Main Courses," teach the reader how to typeset most normally encountered text and mathematics. "Sauces and Pickles," the third section, treats more exotic problems and includes a 60-page dictionary of special TEXniques.

Exercises sprinkled generously through each chapter encourage the reader to sit down at a terminal and learn through experimentation. Appendixes list summaries of frequently used and more esoteric symbols as well as answers to the exercises.



ISBN 0-8218-2999-8, LC 85-7506  
290 pages (softcover), 1986  
AMS Indiv. Memb. \$26, AMS Inst.  
Memb. \$30, List price \$33  
To order specify JOYT/T

Shipping/Handling: 1st book \$2, each  
add'l \$1, max. \$25; by air, 1st book  
\$5, each add'l \$3, max. \$100

**PREPAYMENT REQUIRED.** Order from  
American Mathematical Society  
P. O. Box 1571  
Annex Station  
Providence, RI 01901-1571

or call 800-556-7774 to use VISA or MasterCard.

Prices subject to change.

# THE AMS T<sub>E</sub>X LIBRARY

Your single source for T<sub>E</sub>X products

## • AMS-T<sub>E</sub>X

The AMS macro software that simplifies the typesetting of complex mathematics. AMS-T<sub>E</sub>X supports the use of AMSFonts (below). Available for IBM microcomputers and for Macintosh Plus, SE and II.

## • AMSFonts

AMSFonts (Euler Fraktur, AMS Extra Symbols including Blackboard Bold, Cyrillic Lightface and Bold) are designed for use with AMS-T<sub>E</sub>X. Available Resolutions: 118, 180, 240, and 300 dpi. For use with screen previewers and with drivers for dot matrix and laser printers. When ordering AMSFonts, please specify resolution or type of printer. This information is necessary to process your order. (IBM distribution in standard PK format, Macintosh in *Textures* format)

## • MathSciT<sub>E</sub>X

This macro package is designed to format search output from the database MathSci. MathSciT<sub>E</sub>X is available for MathSci Online and for MathSci Disc, the compact disc (CD-ROM) version of *MR* and *CMP*.

## • The Joy of T<sub>E</sub>X

The Joy of T<sub>E</sub>X is the user-friendly guide to the AMS-T<sub>E</sub>X macro package and details many features of this extremely useful text-processing package. 1986, 290 pages, ISBN 0-8218-2999-8, Softcover

---

### Prices:

AMS-T<sub>E</sub>X: List \$30, AMS Member Price \$27

AMS-T<sub>E</sub>X with Joy of T<sub>E</sub>X: List \$55, AMS Member Price \$50

\*AMS-T<sub>E</sub>X with AMSFonts and MathSciT<sub>E</sub>X: List \$65, AMS Member Price \$59

AMSFonts and MathSciT<sub>E</sub>X: List \$45, AMS Member Price \$41

AMS-T<sub>E</sub>X with AMSFonts, MathSciT<sub>E</sub>X and Joy of T<sub>E</sub>X: List \$90, AMS Member Price \$81

\*Joy of T<sub>E</sub>X: List \$33, AMS Inst Member Price \$30, AMS Indiv Member Price \$26

\*Included at no charge upon request if your order totals \$250 or more. Please specify online or CD-ROM version of MathSciT<sub>E</sub>X.

---

## Also available from the AMS Library of T<sub>E</sub>X Products

The following commercial software may be ordered from the AMS Library – your single source for T<sub>E</sub>X materials. Call or write for prices, (401) 272-9500, ext. 328, or (800) 556-7774, ext. 328 in the continental U.S.; T<sub>E</sub>X Library, American Mathematical Society, PO Box 6248, Providence, RI 02940.

### T<sub>E</sub>X for IBM PC and Compatibles

- PCT<sub>E</sub>X (Personal T<sub>E</sub>X, Inc.) with the PCT<sub>E</sub>X Manual, L<sup>A</sup>T<sub>E</sub>X macros and L<sup>A</sup>T<sub>E</sub>X User's Guide
- pcMF-METAFONT software for PCT<sub>E</sub>X, with manual and METAFONTbook

### T<sub>E</sub>X for Macintosh Plus, SE and II

- Textures* (Blue Sky Research) with built-in screen previewer, ImageWriter/LaserWriter driver and picture embedding capability, and The T<sub>E</sub>Xbook by D. Knuth
- L<sup>A</sup>T<sub>E</sub>X macros with L<sup>A</sup>T<sub>E</sub>X User's Guide

### Printer Drivers

Epson, Okidata, IBM Graphics, Proprinter, Toshiba, HP LaserJet, QMS Lasergrafix, PostScript, Apple LaserWriter, Cordata.

### Screen Previewers for IBM and Compatibles

- Preview (ArborText)
- MAXView (Aurion)
- PTI View (Personal T<sub>E</sub>X, Inc.)

**HOW TO ORDER:** Call the T<sub>E</sub>X Library at (401) 272-9500, or (800) 556-7774 in the continental U.S. to order with VISA or MasterCard. Or write to: T<sub>E</sub>X Library, American Mathematical Society, P. O. Box 6248, Providence, RI 02940. Please add shipping and handling (see below).

PREPAYMENT REQUIRED: Software/Books are sent via UPS Second Day Air to U.S. addresses, first class mail to Canada, and air delivery elsewhere. Add shipping and handling for Software/Books: \$8 per order in the U.S. and Canada; \$35 per order (\$15 for AMS-T<sub>E</sub>X and AMSFonts only) for air delivery outside the U.S. and Canada. Prices subject to change.

Publishing Companion translates  
**WordPerfect**

to

**TEX**

It doesn't take a **TEX**pert to use **TEX**.

With **Publishing Companion**, you can publish documents using **TEX** with **little or no TEX knowledge**. Your WordPerfect files are translated into **TEX** files, so anyone using this simple word processor can immediately begin typesetting their own documents!

And now, K-Talk introduces **Publishing Companion** version 2.0, which translates **WordPerfect 5.0** files into **TEX**.

Other word processors are supported using Mastersoft's WordForWord file conversion utility, \$70.

*Special Introductory Offer*

Retail Price .....	\$249
Academic Discount Price .....	\$199
<b>Introductory Price .....</b>	<b>\$179</b>

This offer good until October 31, 1989. Upgrade from Publishing Companion v.1.XX is \$49.

For the power of **TEX** with the ease of a word processor, **Publishing Companion** is your "best friend" for desktop publishing.

For more information or to place an order, call or write:

**K-TALK**  
 COMMUNICATIONS

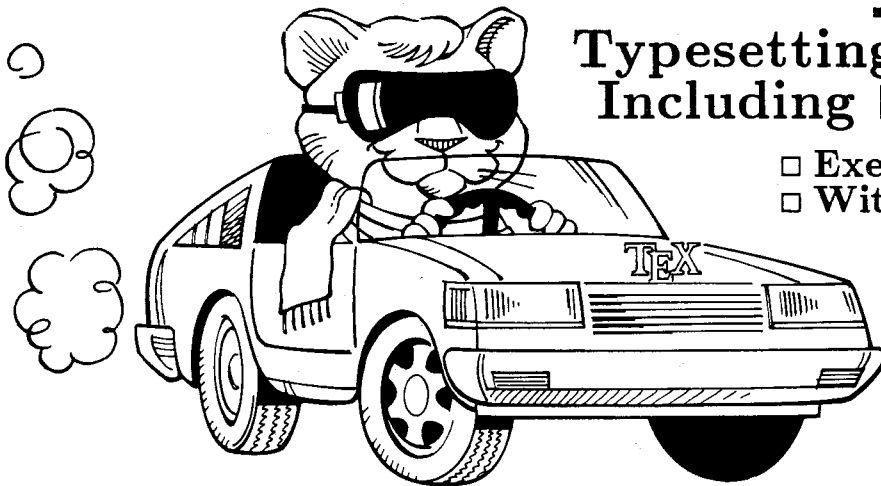
50 McMillen Ave  
 Columbus, Ohio 43201  
 (614) 294-3535

**DESKTOP PUBLISHING HAS NEVER BEEN SIMPLER  
 AND WILL NEVER BE THE SAME**

# TurboTEX

Typesetting Software  
Including METAFONT

- Executables \$150
- With source \$300



TurboTEX Release 2.0 software offers you a complete typesetting package based on the TEX 2.95 and METAFONT 1.7 standards: preloaded plain TEX, L<sup>A</sup>TEX, INITEX, VIRTEX, and plain METAFONT interfaced to CGA/EGA/VGA/Hercules graphics; TRIP and TRAP certification; Computer Modern and L<sup>A</sup>TEX fonts, and printer drivers for HP LaserJet Plus/Series II, PostScript, and dot-matrix printers. New features in the HP LaserJet driver put PCX or TIFF graphics files directly into your TEX documents. This wealth of software fills over 10 megabytes of diskettes, and runs on your IBM PC, UNIX, OS/2, or VAX/VMS system.

■ **Power Features:** TurboTEX brings big-machine performance to your small computer. TurboTEX breaks the 640K memory barrier under MS-DOS on the IBM PC with our virtual memory sub-system. You'll have the same sized TEX that runs on multi-megabyte mainframes, with plenty of memory for large documents, complicated formats, and demanding macro packages that break other TEX implementations. On

larger computers, TurboTEX runs up to 3 times faster in less memory than the Stanford Pascal distribution.

■ **Source code:** Order the TurboTEX source in portable C, and you will receive more disks with over 85,000 lines of generously commented TEX, TurboTEX, METAFONT, and printer driver source code, including: our WEB system in C; PASCHAL, our proprietary Pascal-to-C translator; and preloading, virtual memory, and graphics code. TurboTEX meets C portability standards like ANSI and K&R, and is robustly portable to a growing family of operating systems.

■ **TEX-FAX:** Connects TEX to the FAX revolution. Send perfect TEX output instantly, anywhere, without scanning. With TEX-FAX, any FAX machine in the world becomes your output device! Complete with PC board and software for \$395 (4800 bps) or \$795 (9600 bps).

■ **Desktop Publishing Interface Option:** Converts TEX output (such as equations or tables) for direct use in programs like Ventura Publisher and Pagemaker. (\$50 for PC).

■ **Availability & Requirements:** TurboTEX executables for IBM PC's include the User's Guide and require 640K and hard disk. Order source code (includes Programmer's Guide) for other machines. Source compiles with Microsoft C 5.0 or later on the PC; other systems need 1 MB memory and a C compiler supporting UNIX standard I/O. Media is 360K 5-1/4" PC floppy disks; other formats at extra cost.

■ **No-risk trial offer:** Examine the documentation and run the PC TurboTEX for 10 days. If you are not satisfied, return the software for a 100% refund or credit. (Offer applies to PC executables only.)

#### Ordering TurboTEX

Order 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.

Ask for the free, 50-page Buyer's Guide.

# Kinch

The Kinch Computer Company

PUBLISHERS OF TURBOTEX  
501 South Meadow Street  
Ithaca, New York 14850  
Telephone (607) 273-0222  
FAX (607) 273-0484



## Public Domain T<sub>E</sub>X

The authorized and current versions T<sub>E</sub>X software are available from *Maria Code - Data Processing Services* by special arrangement with Stanford University and other contributing universities. The standard distribution tape contains the source of T<sub>E</sub>X and METAFONT, the macro libraries for A<sub>M</sub>S-T<sub>E</sub>X, L<sub>A</sub>T<sub>E</sub>X, SliT<sub>E</sub>X and HP T<sub>E</sub>X, sample device drivers for a Versetec and LN03 printers, documentation files, and many useful tools.

Since these are in the public domain, they may be used and copied without royalty concerns. They represent the official versions of T<sub>E</sub>X. A portion of your tape cost is used to support development at Stanford University.

If you have a DEC VAX/VMS, IBM CMS, IBM MVS or DEC TOPS operating system, you will want to order a special distribution tape which contains “ready-to-run” T<sub>E</sub>X and METAFONT. If you do not have one of these systems, you must perform a more involved installation which includes compiling the source with your Pascal compiler. Ready-to-run versions of T<sub>E</sub>X are available for other systems from various sources at various prices. You may want to examine these before ordering a standard distribution tape.

The font tapes contain GF files for the Computer Modern fonts. While it is possible to generate these files yourself, it will save you a lot of CPU time to get them on tape.

All systems are distributed on 9 track, 1600 bpi magnetic tapes. If both a distribution tape and a font tape are ordered, they may be combined on a single 2400' reel, space permitting.

Your order will be filled with the current versions of software and manuals at the time it is received. If you want a specific version, please indicate that on your order.

Please use the form on the next page for your order. Note that postage, except domestic book rate is based on the item weights in pounds. If you want to place your order by telephone, please call (408) 735-8006 between 9:00 am and 2:00 pm West Coast time. Do not call for technical assistance since no one there can help you.

We normally have a good stock of books and tapes, so your order can be filled promptly — usually within 48 hours.

Make checks payable to *Maria Code - Data Processing Services*. Export orders must have a check drawn on a US bank or use an International Money Order. Purchase orders are accepted.

## T<sub>E</sub>X Order Form

**T<sub>E</sub>X Distribution tapes:**

- \_\_\_\_\_ Standard ASCII format
- \_\_\_\_\_ Standard EBCDIC format
- \_\_\_\_\_ Special VAX/VMS format Backup
- \_\_\_\_\_ Special DEC 20/TOPS 20 Dumper format
- \_\_\_\_\_ Special IBM VM/CMS format
- \_\_\_\_\_ Special IBM MVS format

**Font Library Tapes (GF files)**

- \_\_\_\_\_ 300 dpi VAX/VMS format
- \_\_\_\_\_ 300 dpi generic format
- \_\_\_\_\_ IBM 3820/3812 MVS format
- \_\_\_\_\_ IBM 3800 CMS format
- \_\_\_\_\_ IBM 4250 CMS format
- \_\_\_\_\_ IBM 3820/3812 CMS format

Tape prices: \$92.00 for first tape,  
\$72.00 for each additional tape.

Total number of tapes \_\_\_\_\_  
Postage: allow 2 lbs. for each tape

**Documents:**

	Price \$	Weight	Quantity
T <sub>E</sub> Xbook (vol. A) softcover .....	27.00	2	_____
T <sub>E</sub> X: The Program (vol. B) hardcover .....	40.00	4	_____
METAFONT book (vol. C) softcover .....	22.00	2	_____
METAFONT: The Program (vol. D) hardcover ..	40.00	4	_____
Computer Modern Typefaces (vol. E) hardcover	40.00	4	_____
L <sup>A</sup> T <sub>E</sub> X document preparation system .....	27.00	2	_____
WEB language * .....	12.00	1	_____
T <sub>E</sub> Xware * .....	10.00	1	_____
BibT <sub>E</sub> X * .....	10.00	1	_____
Torture Test for T <sub>E</sub> X * .....	8.00	1	_____
Torture Test for METAFONT * .....	8.00	1	_____
METAFONTware * .....	15.00	1	_____
Metamarks * .....	15.00	1	_____

\* published by Stanford University

**Payment calculation:**

Number of tapes ordered _____	Total price for tapes _____
Number of documents ordered _____	Total price for documents _____
	Add the 2 lines above _____
Orders from within California: Add sales tax for your location. _____	

**Shipping charges: (for domestic book rate, skip this section)**

Total weight of tapes and books \_\_\_\_\_ lbs.

	_____ domestic priority mail	rate \$1.50/lb.
Check	_____ air mail to Canada and Mexico:	rate \$2.00/lb.
One	_____ export surface mail (all countries):	rate \$1.50/lb.
	_____ air mail to Europe, South America:	rate \$5.00/lb.
	_____ air mail to Far East, Africa, Israel:	rate \$7.00/lb.

Multiply total weight by shipping rate. Enter shipping charges: \_\_\_\_\_

**Total charges:** (add charges for materials, tax and shipping) \_\_\_\_\_

**Send to:** Maria Code, DP Services, 1371 Sydney Drive, Sunnyvale, CA 94087.

Include your name, organization, address, and telephone number.

Are you or your organization a member of TUG? \_\_\_\_\_

If you find yourself needing a *TEX*nician each time you want to create a format for a new document, or modify an existing one, then you need *TEXT1*.

*TEXT1* makes it easy for the *TEX* novice and expert alike to quickly modify the style of your chapters, contents, running head, margin notes, indices, etc. You can get *TEXT1* for a microcomputer for \$150.

*TEXT1* also has an International Phonetic font optionally available for \$100.

For more information, write to:

*TEXT1* Distribution  
WSUCSC  
Pullman, WA 99164-1220



or call:

*TEXT1* Distribution  
509-335-0411

# TEX Made Easy

Using TEX With The Plain Macro Package

## Instruction

Since *TEX Made Easy* was created, author/instructor Daniel Zirin has taught hundreds of new TEX users with this 10 hour seminar format class. Many of the participants were unsophisticated computer users like administrative aids and business people. Mr. Zirin has consistently received high marks in presenting beginning TEX by *TEX Made Easy* students.

Now you can bring *TEX Made Easy* to your place of business at affordable rates.\* Mr. Zirin will present *TEX Made Easy* over a period of 2 to 5 days at your office (unlimited class size). To schedule a two day seminar in most U.S. cities would cost an average \$2500.

Opt. 1\* 10 hours seminar over 2 days (\$1500).

Opt. 2\* 10 hours seminar and 3 hours lab assistance over 2 days (\$1750).

Opt. 7\* 10 hours seminar over 5 days (\$2850).

Opt. 8\* 10 hours seminar and 7½ hours lab assistance over 5 days (\$3325).

\*Restrictions apply. Call or write for details.

Use the money you saved buying TEX teaching the power of TEX!

## Book

*TEX Made Easy*, the book, is written for people interested in a quick **simple** approach to learning how to start using PlainTEX. Written for the business professional, *TEX Made Easy* has been given high praise by those considered "computer shy".

### Topics Covered

- Creating TEX files.
- Defining page layout and margins.
- Accenting and overstriking characters.
- Using different fonts.
- How to use the TEX program.
- Specializing the headline and footline for each page.
- Generating footnotes.
- Setting a tabular environment.
- Using alignment to make tables.
- Dealing with figures and insertions.
- Automatic enumeration of paragraphs, equations, references, etc. . .
- Using math and equation modes.
- Creating simple TEX macros.
- A sample macro package to generate an automatic table of contents.

Price: \$15 for each 90 page book.

\$3 shipping U.S., \$6 all others.

Softcover handbook available soon.

Purchase orders, checks, and money orders accepted.

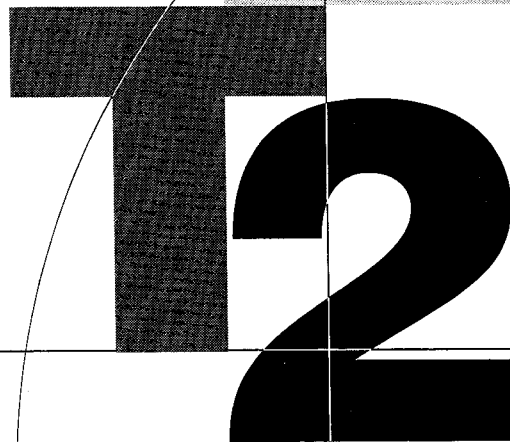
Zar Limited (818) 794-1224  
P. O. Box 372, Pasadena, CA 91102, USA

**T<sub>E</sub>X Device Interfaces for VMS**

**PostScript**

**LaserJet**

**LN03**



**Northlake Software**  
812 SW Washington, Suite 1100  
Portland, Oregon 97205 USA

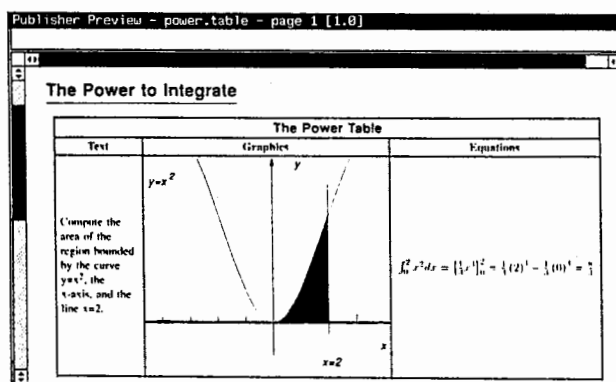
503-228-3383 fax 503-228-5662

The VMS T<sub>E</sub>X specialists



# THE PUBLISHER™

QUALITY PUBLISHING MADE SIMPLE



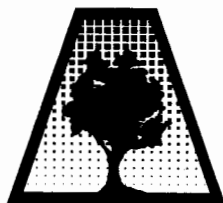
Write a book, a memo... or anything in between.

*THE PUBLISHER* gives you the power, flexibility and quality you've come to expect from  $\text{\TeX}$ , in a window-based, WYSIWYG editing environment. Our Table Editor allows you to create, scale and rule tables containing text, graphics and equations. The Graphics Editors let you create bit-mapped or object-

oriented graphics like a professional. With our Equation Editor, you can write mathematical expressions using the symbols and characters you expect to see on the printed page. All these tools are combined into a single product designed to bring to you the power of  $\text{\TeX}$  and much, much more.

See just how simple quality publishing can be with *THE PUBLISHER*.

Call us today.



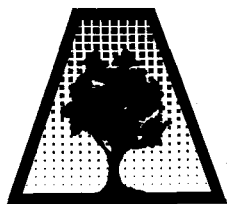
ARBORTEXT INC.

535 West William Street  
Suite 300  
Ann Arbor, MI 48103  
FAX: (313) 996-3573  
**(313) 996-3566**

E-Mail [arbortext@arbortext.com](mailto:arbortext@arbortext.com)

*THE PUBLISHER* runs on Sun workstations and supports SGML, PostScript® and  $\text{\TeX}$  standards.

PostScript is a registered trademark of Adobe Systems, Inc. Sun is a trademark of Sun Microsystems, Inc.  $\text{\TeX}$  is a trademark of the American Mathematical Society.



## WE WERE HERE IN THE BEGINNING

ARBORTEXT INC.

ArborText has been providing quality  $\text{\TeX}$  products from the beginning. We've delivered more  $\text{\TeX}$  products with more features for more platforms than anyone else.

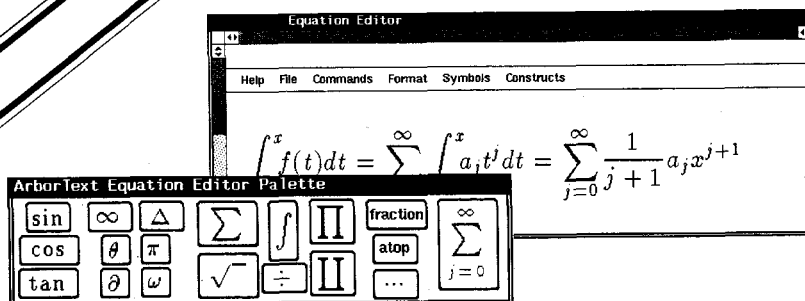
$$\int_0^x f(t) dt = \sum_{j=0}^{\infty} \int_0^x a_j t^j dt = \sum_{j=0}^{\infty} \frac{1}{j+1} a_j x^{j+1}$$

We stand behind our products with knowledgeable customer support, comprehensive documentation and on-going software enhancements.

When you think of  $\text{\TeX}$ , think ArborText.

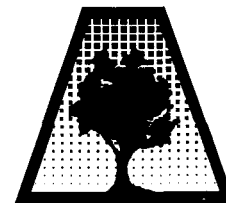
**NEW...  
For Your PC**

*THE EQUATION EDITOR*  
A WYSIWYG editor for writing complex mathematical expressions—coming soon from ArborText.



See our Equation Editor at the  $\text{\TeX}$  Users Group 10th Anniversary Conference

## WE'LL BE HERE TOMORROW.



ARBORTEXT INC.

535 West William Street, Suite 300, Ann Arbor, MI 48103 • (313) 996-3566 • FAX (313) 996-3573

$\text{\TeX}$  is a trademark of the American Mathematical Society.

# Math and Technical Book Publishers . . . .

AMS-TEX  
Computer Modern fonts  
are now available!

If you are creating your files using T<sub>E</sub>X,  
Computer Composition Corporation can now  
offer the following services:

- Converting T<sub>E</sub>X DVI files to fully paginated typeset pages in either "AM" or "CM" fonts.
- Providing 300 dpi laser-printed page proofs which simulate the typeset page in "AM" fonts only.
- Keyboarding services from traditionally prepared manuscripts via the T<sub>E</sub>X processing system in either "AM" or "CM" fonts.
- Full camera work services, including halftones, line art, screens and full-page negatives or positives for your printer.

---

*Call or write for sample pages  
in either "AM" or "CM" fonts*

---

**COMPUTER  
COMPOSITION  
CORPORATION**

1401 WEST GIRARD AVE. • MADISON HEIGHTS, MI 48071

(313) 545-4330



# HAPPY BIRTHDAY TUG!

*To celebrate, Personal T<sub>E</sub>X, Inc. is extending this special offer to all TUG members:  
20% off the regular price of these selected products when ordered before September 15, 1989!  
And on orders of \$300 or more, you'll also get a PC T<sub>E</sub>X T-shirt—free!*

**PTIJET FOR HP DESKJET.** Full featured printer driver for HP DeskJet, PLUS. Laser quality output.  
~~-\$119-~~ \$95

**PC T<sub>E</sub>X + PTILASER + PTIVIEW.** T<sub>E</sub>X82, Version 2.9: professional formatting and typesetting results—for amateur prices. Includes INIT<sub>E</sub>X, L<sub>A</sub>-T<sub>E</sub>X, AMS-T<sub>E</sub>X, VANILLA Macro Pak, PC T<sub>E</sub>X and L<sub>A</sub>T<sub>E</sub>X manuals. Plus a PTILaser device driver, to take full advantage of your laser printer. PLUS the PTIView screen previewer for on-screen previewing of your T<sub>E</sub>X documents and immediate editing. Top performance and low cost make this our most popular package.  
~~-\$499-~~ \$399

**PC T<sub>E</sub>X + PTILASER.** As above, but without the PTIView screen previewer.  
~~-\$399-~~ \$309

**PC T<sub>E</sub>X + PTIDOT + PTIVIEW.** This package gives you all the T<sub>E</sub>X and PTIView benefits, together with our dot-matrix device driver for reliable, low cost printing.  
~~-\$399-~~ \$309

**PC T<sub>E</sub>X + PTIJET + PTIVIEW.** Same as the above package, but with PTIJet instead of PTIDot. Laser quality output.  
~~-\$429-~~ \$343

**PC T<sub>E</sub>X + PTIJET.** As above, without the PTIView screen previewer.  
~~-\$329-~~ \$263

**PCMF—METAFONT for the PC.** Lets you design fonts and create graphics. (Not for the novice.) pcMF Version 1.7.  
~~-\$195-~~ \$156

**PTI LASER HP+, SERIES II.** This device driver for the HP LaserJet Plus and Series II laser printers takes full advantage of the 512K resident memory.  
~~-\$195-~~ \$156

**PTI LASER POSTSCRIPT.** Device driver for PostScript printer; allows the resident fonts and graphic images to be used in T<sub>E</sub>X in documents.  
~~-\$195-~~ \$156

**PTI FONTWARE Interface Package.** Software to generate Bitstream outline fonts at any size. (The Interface is necessary to use Bitstream fonts. Fonts are not included—order below).  
~~-\$95-~~ \$76

**PTI FONTWARE WITH SWISS or DUTCH.** Same as above but includes your choice of either Swiss or Dutch at a special bundled price.  
~~-\$179-~~ \$143

**BITSTREAM Font Families.** An extensive library of 30 type families, in any size you specify, with true typographic quality. Each family: ~~-\$179-~~ \$143

PERSONAL  
  
INC

To order, just dial  
**(415) 388-8853**

12 Madrona Avenue Mill Valley, CA 94941 FAX: (415) 388-8865 VISA, MC accepted.

Requires: DOS 2.0 or later, 512K RAM, 10M hard disk. T<sub>E</sub>X is an American Mathematical Society TM. PC T<sub>E</sub>X is a Personal T<sub>E</sub>X, Inc. TM. Manufacturers' names are their TMs. Outside the USA, order through your local PC T<sub>E</sub>X distributor. Inquire about available distributorships and site licenses. This ad was produced using PC T<sub>E</sub>X and Bitstream fonts.

TYPESETTING: JUST  
**\$2.50**  
PER PAGE!

Send us your T<sub>E</sub>X DVI files and we will typeset your material at 2000 dpi on quality photographic paper — \$2.50 per page!

Choose from these available fonts: Computer Modern, Bitstream Fontware™, and any METAFONT fonts. (For each METAFONT font used other than Computer Modern, \$15 setup is charged. This ad was composed with PCT<sub>E</sub>X® and Bitstream Dutch (Times Roman) fonts, and printed on RC paper at 2000 dpi with the Chelgraph IBX typesetter.)

And the good news is: just \$2.50 per page, \$2.25 each for 100+ pages, \$2.00 each for 500+ pages! Laser proofs \$.50 per page. (\$25 minimum on all jobs.)

Call or write today for complete information, sample prints, and our order form. **TYPE 2000, 16 Madrona Avenue, Mill Valley, CA 94941. Phone 415/388-8873.**

**TYPE**  
**2000**

## The ScripTek Challenge

**We challenged ourselves and now we challenge the competition to offer the T<sub>E</sub>X community a system as comprehensive as our complete System I for less than \$125.**

Simply stated, the ScripTek challenge was to create a total typesetting system that offered value, speed, and flexibility at a price unrivaled by other T<sub>E</sub>X vendors. We met the challenge. We now challenge the others to follow our lead.

### Value

The components of ScripTek's Complete System I can not be purchased separately for less than \$125. The system includes: T<sub>E</sub>X (VIRTEX and INITEX), METAFONT (VIRMF and INIMF) with support for CGA, EGA, VGA and AT&T graphics cards, DVliew (a three zoom level previewer that produces fully scaled fonts on the fly and also supports CGA, EGA, VGA, and AT&T graphics cards), DVlhp (a driver for the HP Laserjet II and compatible laser printers), DVlhp (a driver for the IBM Graphics dot matrix printer and compatible dot matrix printers), all of the CM Fonts including source files, plus TFM, DVI, and GF utility programs like GF to DVI (for full page proofs of METAFONT designs), TF to PL, PL to TF, GF type, and DVI type. That's value.

### Speed

When we translated the mainframe versions of T<sub>E</sub>X and METAFONT into C, we didn't rely upon any convenient shortcuts. Rather, ScripTek translated both these programs one line at a time to achieve the most compact C code possible. The same is true of our virtual memory routines which we incorporated into both T<sub>E</sub>X and METAFONT. These routines, like the screen routines for DVliew, were custom designed for use

with T<sub>E</sub>X and METAFONT, and once developed, were repeatedly profiled under the UNIX operating system. Those functions that were most often called were then rewritten in assembly language to optimize speed and minimize size.

### Flexibility

ScripTek wanted its customers to have options. We designed our installation package to allow you to set dozens of different environment variables that meet your special needs for the right typesetting system. Little touches like routines that automatically call in your favorite editor when you need to make changes are examples of ScripTek's system approach to typesetting. We hand crafted all of the components of the Complete System I to work together in a system that addresses your individual needs.

We don't offer any gimmicks—no newsletters with cute little Gorillas on the cover—just the finest quality typesetting for serious scientific and literary work at the lowest price. Even the call is free. For phone orders, call 1-800-950-1998 and listen for the dial tone. Then dial 383-5022 (must use touch-tone phone). We accept MasterCard and Visa, and our operators are there 24 hours a day, seven days a week. If you wish to order by mail with a personal check, or for more information write: ScripTek, Inc., Sales Department, P.O. Box 5022, Kansas City, Missouri 64131.

Requires IBM PC XT/AT or compatible, 512K or more RAM, and MS DOS operating system. Postage, handling and tariffs extra.

*Value, Speed, Flexibility. The ScripTek Challenge.*



## Publishing Services



### From the Basic

The American Mathematical Society can offer you a basic T<sub>E</sub>X publishing service. You provide the DVI file and we will produce typeset pages using an Autologic APS Micro-5 phototypesetter. The low cost is basic too: only \$5 per page for the first 100 pages; \$2.50 per page for additional pages, with a \$30 minimum. Quick turnaround is important to you and us ... a manuscript up to 500 pages can be back in your hands in just one week or less.

### To the Complex

As a full service T<sub>E</sub>X publisher, you can look to the American Mathematical Society as a single source for all your publishing needs.

Macro-Writing	T <sub>E</sub> X Problem Solving	Autologic Fonts	Keyboarding
Art and Pasteup	Camera Work	Printing	Binding

For more information or to schedule a job, please contact Regina Girouard, American Mathematical Society, P.O. Box 6248, Providence, RI 02940 or call 401-272-9500 or 800-556-7774 in the continental U.S.

**NEW FROM:**

The Coolspring Banjo Works  
6617 Home Road  
Delaware, OH 43015  
U.S.A.

METAPLOT

**\$50.**  
postpaid \*

Use your favorite computer aided drafting program to communicate graphically with METAFONT. Create your own dingbats, borders, line illustrations, and logos for T<sub>E</sub>X printing on all systems and devices.

\* Outside North America? Please write for postage rates and overseas distributors. Ohio residents please add 6% sales tax.

**Ordering Information:**  
3.5 Inch Diskette Format (check one):  
Amiga  MS-DOS  Atari ST   
(Other? We'll try! \_\_\_\_\_)  
Computer System \_\_\_\_\_  
Graphics Software \_\_\_\_\_

## THE WRITE STUFF, TEXNICALLY SPEAKING

### T<sub>E</sub>X is the write stuff

*T<sub>E</sub>X is the powerful publishing system that is guaranteed to make any document you write easier to read... and guaranteed to make that document say the right stuff about you!*

*Micro Programs, Inc. is your source for T<sub>E</sub>X and related ArborText products for IBM PC and Sun workstations.*

*Call Bob Harris on (516) 921-1351 and get the name of the dealer nearest you.*

**MICRO PROGRAMS, INC.**  
251 JACKSON AVENUE  
SYOSSET  
NY 11791

# **T<sub>E</sub>X Users Group Membership List — Supplement**

July 1989

This supplementary list, compiled on 9 June 1989, includes the names of all persons who have become members of TUG or whose addresses have changed since publication of the last full membership list, as of 3 March 1989. Total membership: 151 institutional members and 3,268 individuals affiliated with more than 1,350 colleges and universities, commercial publishers, government agencies, and other organizations throughout the world having need for an advanced composition system.

The following information is included for each listing of an individual member, where it has been provided:

- Name and mailing address
- Telephone number
- Network address
- Title and organizational affiliation, when that is not obvious from the mailing address
- Computer and typesetting equipment available to the member, or type of equipment on which his organization wishes to (or has) installed T<sub>E</sub>X
- Uses to which T<sub>E</sub>X may be put, or a general indication of why the member is interested in T<sub>E</sub>X

## CONTENTS

Board of Directors	2
Site Coordinators, TUG Committees	3
Addresses of TUG Members: additions and changes from 3 March 1989 through 9 June 1989	4
T <sub>E</sub> X consulting and production services for sale	11

Recipients of this list are encouraged to use it to identify others with similar interests, and, as TUG members, to keep their own listings up-to-date in order for the list to remain as useful as possible. New or changed information may be submitted on the membership renewal form bound into the back of a recent issue of TUGboat. Comments on ways in which the content and presentation of the membership list can be improved are welcome.

This list is intended for the private use of TUG members; it is not to be used as a source of names to be included in mailing lists or for other purposes not approved by TUG. Additional copies are available from TUG. Mailing lists of current TUG membership are available for purchase. For more information, contact Ray Goucher, TUG Executive Director.

Distributed with TUGboat Volume 10 (1989), No. 2. Published by

**T<sub>E</sub>X Users Group**  
P. O. Box 9506  
Providence, R.I. 02940-9506, U.S.A.

## TUG Board of Directors

## Officers

**CHILDS, S. Bart**

Department of Computer Science  
Texas A & M University  
College Station, TX 77843-3112  
409-845-5470

bart@cssun.tamu.edu

Bitnet: Bart@TAMLSR

President; Finance Committee;  
DG MV Site Coordinator

**FURUTA, Richard**

Department of Computer Science  
University of Maryland  
College Park, MD 20742  
301-454-1461

furuta@mimsy.umd.edu

Vice President; Finance Committee;  
Laser-Lovers moderator

**HOENIG, Alan**

17 Bay Avenue  
Huntington, NY 11743  
516-385-0736

Secretary; Finance Committee

**NESS, David**

803 Mill Creek Road  
Gladwyne, PA 19035  
215-649-3474

Treasurer; Finance Committee

## Other Board Members

**BEETON, Barbara**

American Mathematical Society  
P. O. Box 6248  
Providence, RI 02940

401-272-9500 x299

BNB@Math.AMS.com

TUGboat@Math.AMS.com

Editor, TUGboat

**CARNES, Lance**

163 Linden Lane  
Mill Valley, CA 94941  
415-388-8853

Site Coordinator for "small" systems

**CLARK, Malcolm W.**

Imperial College  
Computer Center  
Exhibition Rd  
London SW7 2BP, England  
01-589-5111 x 4949  
Unix: mwc@doc.ic.ac.uk

**CRAWFORD, John M.**

Computing Services Center  
College of Business  
Ohio State University  
1775 College Road  
Columbus, OH 43210  
614-292-1741  
crawford-j@osu-20.ircc.ohio-state.edu  
Bitnet: TS0135@OHSTVM  
Prime 50 Series Site Coordinator

**DYER, Allen R.**

13320 Tridelphia Road  
Ellicott City, MD 21043  
(301) 531-3965

**FARRELL, Shawn**

Computing Centre  
McGill University  
805 Sherbrooke St W  
Montréal H3A 2K6, Québec Canada  
514-398-3676  
Bitnet: CCSF@MCGILLA

**FOX, Jim**

Academic Computing Center HG-45  
University of Washington  
3737 Brooklyn Ave NE  
Seattle, WA 98105  
206-543-4320  
fox@uwvm.acs.washington.edu  
Bitnet: fox7632@uwacdc  
CDC Cyber Site Coordinator

**FUCHS, David**

1775 Newell  
Palo Alto, CA 94303  
415-323-9436

**GIROUARD, Regina**

American Mathematical Society  
P. O. Box 6248  
Providence, RI 02940  
401-272-9500 x224  
RMG@Math.AMS.com

**GOUCHER, Raymond**

TEX Users Group  
P. O. Box 9506  
Providence, RI 02940-9506  
401-751-7760  
REG@Math.AMS.com  
Executive Director

**GUENTHER, Dean**

Computing Service Center  
Washington State University  
Pullman WA 99164-1220  
509-335-0411  
Bitnet: Guenther@WSUVM1  
Finance Committee; IBM VM/CMS  
Site Coordinator; Annual Meeting  
Program Coordinator

**HAMILTON, Hope**

National Center for  
Atmospheric Research  
P. O. Box 3000  
Boulder, CO 80307  
303-497-8915  
Hamilton@MMM.UCAR.Edu

**HENDERSON, Doug**

Division of Library Automation  
Office of the President  
University of California, Berkeley  
300 Lakeside Drive, Floor 8  
Oakland, CA 94612-3550  
415-987-0561  
Bitnet: dlatex@ucbmsa  
Metafont Coordinator

**ION, Patrick D.**

Mathematical Reviews  
416 Fourth Street  
P. O. Box 8604  
Ann Arbor, MI 48107  
313-996-5273  
ion@Math.AMS.com

**KELLERMAN, David**

Northlake Software  
812 SW Washington  
Portland, OR 97205  
503-228-3383  
Usenet: imagen@negamildavek  
VAX (VMS) Site Coordinator

**KNUTH, Donald E.**

Department of Computer Science  
Stanford University  
Stanford, CA 94305  
DEK@Sail.Stanford.edu

**KRATZER, David H.**

Los Alamos National Laboratory  
P. O. Box 1663, C-10 MS B296  
Los Alamos, NM 87545  
(505) 667-2864  
dhk@lanl.gov

**MacKAY, Pierre A.**

Northwest Computer Support Group  
University of Washington  
Mail Stop DW-10  
Seattle, WA 98195  
206-543-6259  
MacKay@June.CS.Washington.Edu  
UNIX Site Coordinator

**PLATT, Craig R.**

Dept of Math & Astronomy  
Machray Hall  
University of Manitoba  
Winnipeg R3T 2N2, Manitoba  
Canada  
204-474-9832  
Bitnet: platt@uofmcc  
Bitnet: platt@ccm.UManitoba.CA  
CSnet: platt@uofm.cc.cdn  
IBM MVS Site Coordinator

**THIELE, Christina**

Canadian Journal of Linguistics  
Carleton University  
Ottawa K1S 5B6, Ontario Canada  
Bitnet: WSSCAT@CARLETON

**WHIDDEN, Samuel B.**

American Mathematical Society  
P. O. Box 6248  
Providence, RI 02940  
401-272-9500  
sbw@Math.AMS.com  
Finance Committee

**ZAPF, Hermann**

Seitersweg 35  
D-6100 Darmstadt  
Federal Republic of Germany

Addresses and telephone numbers of individuals serving in more than one capacity are listed only once. Unless indicated otherwise, network address are for the Internet.

## Site Coordinators

- **CDC Cyber**  
**FOX, Jim**  
 Academic Computing Center HG-45  
 University of Washington  
 3737 Brooklyn Ave NE  
 Seattle, WA 98105  
 206-543-4320  
 fox@uwavm.acs.washington.edu  
 Bitnet: fox7632@uwacdc  
 CDC Cyber Site Coordinator
- **DG MV**  
**CHILDS, S. Bart**  
 Dept of Computer Science  
 Texas A & M University  
 College Station, TX 77843-3112  
 409-845-5470  
 bart@cssun.tamu.edu  
 Bitnet: Bart@TAMLSR  
 President; Finance Committee
- **IBM MVS**  
**PLATT, Craig R.**  
 Dept of Math & Astronomy  
 Machray Hall  
 University of Manitoba  
 Winnipeg R3T 2N2, Manitoba  
 Canada  
 204-474-9832  
 Bitnet: platt@uofmcc  
 Bitnet: platt@ccm.UManitoba.CA  
 CSnet: platt@uofm.cc.cdn
- **IBM VM/CMS**  
**GUENTHER, Dean**  
 Computing Service Center  
 Washington State University  
 Pullman WA 99164-1220  
 509-335-0411  
 Bitnet: Guenther@WSUVM1  
 Annual Meeting Program Coordinator
- **Prime 50 Series**  
**CRAWFORD, John M.**  
 Computing Services Center  
 College of Business  
 Ohio State University  
 1775 College Road  
 Columbus, OH 43210  
 614-292-1741  
 Crawford-J@Ohio-State.edu  
 Bitnet: TS0135@OHSTVMA  
 International Coordinator
- **"small" systems**  
**CARNES, Lance**  
 163 Linden Lane  
 Mill Valley, CA 94941  
 415-388-8853
- **UNIX**  
**MacKAY, Pierre A.**  
 Northwest Computer Support Group  
 University of Washington  
 Mail Stop DW-10  
 Seattle, WA 98195  
 206-543-6259  
 MacKay@June.CS.Washington.Edu
- **VAX (VMS)**  
**KELLERMAN, David**  
 Northlake Software  
 812 SW Washington  
 Portland, OR 97205  
 503-228-3383  
 Usenet: imagen@negamildavek

## Committees

- **Annual Meeting Program Committee**  
**GUENTHER, Dean R.**  
 509-335-0411  
 Bitnet: GUENTHER@WSUVM1  
 (Board of Directors)  
 Annual Meeting Program Coordinator
- **Committee on Local User Groups**  
**CLARK, Malcolm W.**  
 Unix: mwc@doc.ic.ac.uk  
 (Board of Directors)
- **Output Device Standards Committee**  
**McGAFFEY, Robert W.**  
 Martin Marietta Energy Systems, Inc.  
 Building 9104-2  
 P. O. Box Y  
 Oak Ridge, TN 37831  
 615-574-0618  
 McGaffey%ORN.MFEnet@nmfccc.arpa
- **1989 Scholarship Committee**  
**SHARLOW, Larry**  
 10 Toltec #3  
 Flagstaff, AZ 86001  
 602-774-1630
- **TUGboat Editorial Committee**  
**BEETON, Barbara**  
 TUGboat@Math.AMS.Com  
 (Board of Directors)  
 Editor
- **DAMRAU, Jackie**  
 Mission Research Corporation  
 1720 Randolph Road SE  
 Albuquerque, NM 87106-4245  
 505-768-7647  
 damrau@dbitch.unm.edu  
 Bitnet: damrau@bootes  
 Associate Editor, L<sup>A</sup>T<sub>E</sub>X
- **HOENIG, Alan**  
 (Board of Directors)  
 Associate Co-Editor, Typesetting on  
 Personal Computers
- **HOSEK, Don**  
 3916 Elmwood  
 Stickney, IL 60402  
 Bitnet: U33297@UICVM  
 Associate Editor for Output Devices
- **HAMILTON, Hope**  
 (Board of Directors)
- **HENDERSON, Doug**  
 Bitnet: dlatex@ucbmsa  
 (Metafont Coordinator)
- **HENDERSON, Doug**  
 415-642-9485  
 Bitnet: dlatex@ucbmsa  
 (Board of Directors)
- **THIELE, Christina**  
 Bitnet: WSSCAT@CARLETON  
 (Board of Directors)
- **THIELE, Christina**  
 Bitnet: WSSCAT@CARLETON  
 (Board of Directors)
- **Membership Committee**  
**LAURENDEAU, Charlotte V.**  
 T<sub>E</sub>X Users Group  
 P. O. Box 9506  
 Providence, RI 02940-9506  
 401-751-7760  
 (ex officio)
- **Nominating Committee**  
**BARNHART, Elizabeth**  
 National EDP Department  
 TV Guide  
 Radnor, PA 19088  
 215-293-8890
- **MacKAY, Pierre A.**  
 Department of Computer Science,  
 FR-35  
 University of Washington  
 Seattle, WA 98195  
 206-543-6259  
 MacKay@June.CS.Washington.Edu  
 (Board of Directors)
- **BEETON, Barbara**  
 401-272-9500 x299  
 BNB@Math.AMS.com  
 (Board of Directors)
- **DIYER, Allen R.**  
 13320 Tridelphia Road  
 Ellicott City, MD 21043  
 (301) 531-3965  
 (Board of Directors)  
 Chair
- **GOUCHER, Raymond**  
 401-751-7760  
 reg@Math.AMS.com  
 (Board of Directors)  
 (ex officio)
- **KNUTH, Donald E.**  
 (Board of Directors)  
 (ex officio)
- **PRICE, Lynne A.**  
 Hewlett-Packard  
 3200 Hillview Avenue  
 Palo Alto, CA 94304  
 408-857-4075
- **WHIDDEN, Samuel B.**  
 401-272-9500  
 sbw@Math.AMS.com  
 (Board of Directors)
- **JÜRGENSEN, Helmut**  
 Dept of Computer Science  
 Univ of Western Ontario  
 London N6A 5B7, Ontario, Canada  
 519-661-3560  
 Bitnet: A505@UWOCC1  
 UUCP: helmut@depthot  
 Associate Editor for Software
- **MANN, Laurie D.**  
 Stratus Computer  
 55 Fairbanks Blvd  
 Marlboro, MA 01752  
 617-460-2610  
 uucp: harvard@lanvilles!Mann  
 Associate Editor for Training issues
- **PFEFFER, Mitch**  
 Suite 90  
 148 Harbor View South  
 Lawrence, NY 11559  
 516-239-4110  
 Associate Co-Editor, Typesetting on  
 Personal Computers
- **TOBIN, Georgia K.M.**  
 The Metafoundry  
 OCLC Inc., MC 485  
 6565 Frantz Road  
 Dublin, OH 43017  
 614-764-6087  
 Associate Editor for Fonts
- **WHITNEY, Ron**  
 T<sub>E</sub>X Users Group  
 P. O. Box 9506  
 Providence, RI 02940  
 TUGboat@Math.AMS.com  
 Production Editor

# TEX USERS GROUP

Have you ever wished you knew how to use the full potential of TEX or L<sup>A</sup>TEX properly?

Can't get away to take a course? Let us help you!

Here's your opportunity to learn more about TEX.

If you know of six to fifteen individuals interested in learning more about TEX or L<sup>A</sup>TEX, we'll come to you!\* Check around your own organization; check with other organizations in your locality. (If you have less than six, let us know; we may be able to combine two or more small groups.)

NAME OF CONTACT/TELEPHONE: \_\_\_\_\_

Names of individuals interested in courses	Dept./Co.	Level of Instruction

Please check one:

- We have a training facility equipped with computer terminals, PC's or Macs.  
 We do not have a training facility.

Notes:

Completion of this inquiry does not obligate you in any way.

Fees will vary depending on the course offered and the number of participants.

Some of the courses we offer are:

- TEX: all levels, macro writing, output routines, wizard
- L<sup>A</sup>TEX: all levels, style files
- METAFONT
- PostScript

If you would like to have detailed descriptions of the courses we offer or need additional information, contact Charlotte Laurendeau at the TUG office:

P. O. Box 9506  
 Providence, RI 02940, U.S.A.  
 401-751-7760

In addition to having conducted these courses over the past several years throughout the U. S. and Europe as part of our Regional Course program, we have offered courses on-site for a number of organizations, including:

American Mathematical Society,  
 Beckman Instruments, Inc.,  
 Brookhaven National Lab,  
 Digital Equipment Corp.,  
 Digital Equipment Corp. (England),  
 Institute for Advanced Study (Princeton),  
 Lawrence Livermore National Lab,  
 Lawrence Berkeley National Lab,  
 Los Alamos National Lab,  
 Rutgers University,  
 Science Applications (Las Vegas),  
 University of Delaware,  
 University of Washington, and  
 Woods Hole Oceanographic Institute.

In many cases TUG has conducted several different courses for each these organizations, especially Los Alamos National Lab, where more than two dozen courses at all levels of TEX and L<sup>A</sup>TEX have been conducted since 1985.

\*Based on availability of a properly equipped training facility.



## TeX Consulting and Production Services

### North America

#### AMERICAN MATHEMATICAL SOCIETY

P. O. Box 6248, Providence, RI 02940;  
(401) 272-9500, ext. 224

Typesetting from DVI files on an Autologic APS Micro-5.  
Times Roman and Computer Modern fonts.  
Composition services for mathematical and technical  
books and journal production.

#### ARBORTEXT, Inc.

535 W. William, Suite 300, Ann Arbor, MI 48103;  
(313) 996-3566

Typesetting from DVI files on an Autologic APS-5.  
Computer Modern and standard Autologic fonts.  
TeX installation and applications support.  
TeX-related software products.

#### ARCHETYPE PUBLISHING, Inc.,

Lori McWilliam Pickert

P. O. Box 6567, Champaign, IL 61821; (217) 359-8178  
Experienced in producing and editing technical journals  
with TeX; complete book production from manuscript  
to camera-ready copy; TeX macro writing including  
complete macro packages; consulting.

#### HOENIG, Alan

17 Bay Avenue, Huntington, NY 11743; (516) 385-0736  
TeX typesetting services including complete book  
production; macro writing; individual and group  
TeX instruction.

#### KUMAR, Romesh

1549 Ceals Court, Naperville, IL 60565; (312) 972-4342  
Beginners and intermediate group/individual instruction  
in TeX. Development of TeX macros for specific  
purposes. Using TeX with FORTRAN for  
custom-tailored software. Flexible hours, including  
evenings and weekends.

#### OGAWA, Arthur

920 Addison, Palo Alto, CA 94301; (415) 323-9624  
Experienced in book production, macro packages,  
programming, and consultation. Complete book  
production from computer-readable copy to  
camera-ready copy.

#### RICHERT, Norman

1614 Loch Lake Drive, El Lago, TX 77586;  
(713) 326-2583

TeX macro consulting.

#### TECHNOLOGY, Inc., Amy Hendrickson

57 Longwood Ave., Brookline, MA 02146;  
(617) 738-8029.

TeX macro writing (author of MacroTeX); custom macros  
written to meet publisher's or designer's specifications;  
instruction.

---

### Outside North America

#### BAZARGAN, Kaveh

Optics Section, Blackett Laboratory, Imperial College  
of Science and Technology, London SW7 2BZ, U.K.;  
(01) 589 5111 ext. 6841

Instruction in TeX, for beginner and intermediate levels.  
Custom macros, including complex tables.

#### TECHWORKS Pty. Ltd.

78 Nott Street, Port Melbourne, Victoria 3207, Australia;  
61 3 646 5613; Fax: 61 3 6463550

Commercial versions of TeX and screen previewers for  
computers including PC, Macintosh, Amiga, Unix,  
VAX, and TeX-related products; print drivers for dot  
matrix printers, 300 dpi lasers, and phototypesetters  
(Autologic, Compugraphic and Linotronic);  
assistance with information on public domain software;  
consulting for typesetters and other TeX users.