

... to Don Knuth, that premier computer scientist . . . ,  
we are indebted for T<sub>E</sub>X, which deserves simultaneous  
recognition as the text formatter of choice and the most  
idiosyncratic programming language known to us.

Stephen A. Ward and  
Robert H. Halstead, Jr.  
*Computation Structures* (1990)

# TUGBOAT

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

VOLUME 11, NUMBER 2 • JUNE 1990  
PROVIDENCE • RHODE ISLAND • U.S.A.

## **TUGboat**

During 1990, the communications of the TeX 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. 11, No. 4, is September 11, 1990; the issue will be mailed in October. (Deadlines for future issues are listed in the Calendar, page 308.)

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 TeX or LATEX, 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 Karen Butler at the TUG office.

## **TUGboat Editorial Committee**

Barbara Beeton, *Editor*

Ron Whitney, *Production Assistant*

Helmut Jürgensen, *Associate Editor, Software*

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

Don Hosek, *Associate Editor, Output Devices*

Jackie Damrau, *Associate Editor, LATEX*

Alan Hoenig and Mitch Pfeffer, *Associate Editors, Typesetting on Personal Computers*

*See page 151 for addresses.*

### **Other TUG Publications**

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

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

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

DOS and MS/DOS are trademarks of MicroSoft Corporation

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

METAFONT is a trademark of Addison-Wesley Inc.  
PC TeX is a registered trademark of Personal TeX, Inc.

PostScript is a trademark of Adobe Systems, Inc.  
TeX and AMS-TeX are trademarks 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**

**Karen Butler**

**Charlotte Laurendeau**

P. O. Box 9506  
Providence, RI 02940-9506  
or

653 North Main Street  
Providence, RI 02904  
401-751-7760  
Fax: 401-751-1071  
**TUG@Math.AMS.com**

**Elizabeth Barnhart**

National EDP Dept  
TV Guide  
#4 Radnor Corporate Center  
Radnor, PA 19088  
215-293-8890

**Nelson H. F. Beebe**

Center for Scientific Computing and  
Department of Mathematics  
South Physics Building  
University of Utah  
Salt Lake City, UT 84112  
801-581-5254  
**beebe@science.utah.edu**

**Barbara Beeton**

American Mathematical Society  
P. O. Box 6248  
Providence, RI 02940  
401-455-4014  
**bnn@Math.AMS.com**  
**TUGboat@Math.AMS.com**

**Gerhard Berendt**

Institut für Mathematik I  
Freie Universität Berlin  
Arnimallee 2-6  
1000 Berlin 33  
Germany  
**berendt@fubinf.uucp**

**Janusz S. Bień**

Institute of Informatics  
Warsaw University  
PKiN p.850  
00-901 Warszawa, Poland

**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**  
Superconducting Supercollider  
Laboratory  
Stoneridge Office Park, Suite 260  
250 Beckleymeade Avenue  
Dallas, TX 75237  
214-708-6048  
**damrau@sscvx1.ssc.gov**  
Bitnet: **damrau@sscvx1**

**Georg Denk**  
Mathematisches Institut  
Technische Universität München  
Arcisstraße 21  
D-8000 München 2  
Bitnet: **T1111AA@DMOLRZ01**

**Luzia Dietsche**  
Rechenzentrum der Univ. Heidelberg  
Im Neuenheimer Feld 293  
D-6900 Heidelberg 1  
Federal Republic of Germany  
Bitnet: **X68@DHURZ1**

**Allen R. Dyer**  
13340 Hunt Ridge  
Ellicott City, MD 21043  
301-531-3965

**Victor Eijkhout**  
Department of Mathematics  
University of Nijmegen  
Toernooiveld 5  
6525 ED Nijmegen  
The Netherlands  
080-613169  
Bitnet: **U641001@HNYKUN11**

**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@cs.umd.edu**

**Bernard Gaulle**  
91403 Orsay Cedex, France  
Bitnet: **UCIR001@FRORS31**

**Regina Girouard**  
American Mathematical Society  
P. O. Box 6248  
Providence, RI 02940  
401-455-4000  
**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**

**Roswitha Graham**  
K.T.H. Royal Institute of Technology  
DAB  
100 44 Stockholm, Sweden  
46 (08) 7906525  
uucp: **roswitha@admin.kth.se**

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

**Khanh Ha**  
14912 Village Gate Drive  
Silver Spring, MD 20906

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

**Doug Henderson**  
Blue Sky Research  
534 SW Third Ave  
Portland, OR 97204  
800-622-8398; 503-222-9571;  
TLX 9102900911

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

**Don Hosek**  
 Platt Campus Center  
 Harvey Mudd College  
 Claremont, CA 91711  
 DHosek@HMCVAX.Claremont.Edu

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

**Calvin W. Jackson, Jr.**  
 1749 Micheltorena St.  
 Los Angeles, CA 90026  
 818-356-6245  
 CALVIN@CSVAX.Caltech.Edu

**Alan Jeffrey**  
 Programming Research Group  
 Oxford University  
 11 Keble Road  
 Oxford OX1 3QD  
 Alan.Jeffrey@uk.ac.oxford.prg

**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  
 Fax: 503-228-5662  
 uucp: uunet!nls!davek

**Donald E. Knuth**  
 Department of Computer Science  
 Stanford University  
 Stanford, CA 94305

**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

**C.G. van der Laan**  
 Rekencentrum  
 Universiteit Groningen  
 WSN-gebouw Postbus 800  
 9700 AV Groningen  
 The Netherlands  
 Bitnet: CGL@HGRRUG5

**Joachim Lammarsch**  
 Research Center  
 Universität Heidelberg  
 Im Neuenheimer Feld 293  
 6900 Heidelberg  
 Federal Republic of Germany  
 Bitnet: X92@DHDURZ1

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

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

**Frank Mittelbach**  
 Electronic Data Systems  
 (Deutschland) GmbH  
 Eisenstraße 56  
 D-6090 Rüsselsheim  
 Federal Republic of Germany  
 Bitnet: pzf5hz@drueds2

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

**Ted Nieland**  
 Control Data Corporation  
 Suite 200  
 2970 Presidential Drive  
 Fairborn, OH 45324  
 TNIELAND@AMRL.AF.MIL

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

**Lee S. Pickrell**  
 Wynne-Manley Software, Inc.  
 1094 Big Rock Loop  
 Los Alamos, NM 87544  
 pickrell%lsn.mfenet@ccc.nmfcc.gov

**Craig Platt**  
 Department of Math & Astronomy  
 Machray Hall  
 University of Manitoba  
 Winnipeg R3T 2N2, Manitoba, Canada  
 204-474-9832  
 platt@ccm.UManitoba.CA  
 Bitnet: platt@uofmcc

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

**Rainer Schöpf**  
 Institut für Theoretische Physik der  
 Universität Heidelberg  
 Philosophenweg 16  
 D-6900 Heidelberg  
 Federal Republic of Germany  
 Bitnet: BK4@DHDURZ1

**Len Schwer**  
 APTEK, Inc.  
 4320 Stevens Creek Blvd.  
 Suite 195  
 San Jose, CA 95129  
 micro2.schwer@sri.com

**Philip Taylor**  
 The Computer Centre  
 Royal Holloway and Bedford New  
 College  
 University of London  
 Egham, Surrey TW20 0EX, England  
 P.Taylor@Vax.Rhbnc.Ac.UK

**Christina Thiele**  
 Journal Production Centre, DT1711  
 Carleton University  
 Ottawa K1S 5B6, Ontario Canada  
 613-788-2340  
 Bitnet: WSSCAT@Carleton.CA

**Georgia K.M. Tobin**  
 The Metafoundry  
 OCLC Inc., MC 485  
 6565 Frantz Road  
 Dublin, OH 43017  
 614-764-6087

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

**Ken Yap**  
 Department of Computer Science  
 University of Rochester  
 ken@cs.rochester.edu

**Hermann Zapf**  
 Seitersweg 35  
 D-6100 Darmstadt  
 Federal Republic of Germany

## General Delivery

### Editorial Comments

Barbara Beeton

### Comings and goings

The TUG Board of Directors has recently received the following message from Jim Fox, erstwhile Site Coordinator for CDC Cyber:

As I no longer have a CDC Cyber with which to work, and have not had any requests for Cyber TeX in a couple of years, I feel that I can no longer be considered the CDC Cyber site coordinator for TUG.

In the same message, Jim resigned from the Board as well. I'd just like to thank him publicly for his efforts in TUG's behalf during his tenure.

Shawn Farrell has also resigned from the Board, explaining that he had left McGill for a new job. Shawn was largely responsible for the local arrangements for the Montréal meeting in 1988, a most enjoyable event. Thanks to you too, Shawn, and best wishes for success in your new job.

### Challenges

I find several thought-provoking comments on weaknesses in the support structure for TeX in Liz Barnhart's summary of responses to a questionnaire on the experiences of TeX users in production environments (see below; the questionnaire appeared in *TUGboat* 9, no. 2).

It appears that users feel largely on their own when it comes to learning TeX, solving problems and searching for support. This is the down side of TeX's status as public domain software. Everyone expects to pay, sometimes quite large sums, for proprietary software, and for associated training and support. But for "free" software, no matter how complicated or how high the quality, it is somehow expected that the price of training and support will be likewise very low in price. In fact, there should be room for both options: low-cost but time-intensive, and ready-made but for a price.

**Opportunities for volunteers.** A middle ground exists because volunteers are willing to help out. While access to volunteers is relatively available

over the electronic networks, many TeX users aren't fortunate enough to have network access. And, as the number of good personal computer implementations of TeX increases, the number of "isolated" users is likely to increase as well. Liz has mentioned a local group that she helped to organize, and that is a useful approach. (The TUG office may be able to help; get in touch with Ray Goucher or Charlotte Laurendeau.)

**A place for consultants.** But volunteer activity doesn't really solve the problem of how to develop major new applications. Although the number of self-help guides and similar publications is increasing, tackling such a project means that you must either master TeX yourself, or find help. If the project schedule doesn't permit time for your education, an inquiry to the usual sources (TeXhax et al.) doesn't yield any leads, and there isn't a good local source of TeX talent, then it may be advisable to obtain the services of a consultant. This will cost money, of course, but for any project of substantial size, it may cost less and will almost certainly take less time than trial and error. If you are prepared with a complete and precise statement of the specs for the job, a firm schedule, and a determination to make as few changes as possible after work has started, you will not only minimize the cost, but gain the consultant's respect, and willingness to work with you again. And, if you require as part of the initial specs that the macro package written for your project include thorough documentation, then you also have the opportunity of furthering your education by studying it. The TUG office keeps a list of consultants who can be called on when an inquiry involves more than a quick answer; the list is also published with every edition or supplement of the membership list, and qualified additions are always welcome.

**Help for beginners — a plea.** For those users who are just starting out, and really want to learn TeX well, the legitimate criticism has been made that *TUGboat* contains very little material intended for beginners. To this criticism I respond that *TUGboat* can only publish what is submitted, and there seem to be few aspiring authors who are writing for this audience. If you happen to be such a person, please prove me wrong, and send in your contribution.

---

## **TeX in the Production Environment— Questionnaire Responses**

Elizabeth M. Barnhart

### **Why the Questionnaire?**

Over a year ago I put a questionnaire in *TUGboat* 9, no. 2 asking non-academic users about their dealings with *TeX* and many related aspects. I was interested in finding out what problems other *TeX* users operating in a production environment had, and how they solved those problems.

First I have to start by thanking the people who took the time to return the questionnaire. The responses came from all over the world—17 states as well as Australia, Canada, Denmark, France, Finland, Great Britain, Israel, the Netherlands, Singapore, Spain, Switzerland, and West Germany—and gave quite a variety of “flavors” of *TeX* use, and problems.

Response was better than expected, with a total of 60 people returning questionnaires. Some people are so much into what they can do with *TeX* that a number of them even sent samples of output. Of course there were people who set up their responses in *TeX*, even using the “check mark” from the math font to mark their responses.

Responses went from intense hatred to complete infatuation with *TeX*. As you read the responses, you will find that some of the feedback contradicts other answers (“One man’s meat is another man’s poison”), and other answers were obviously made because the user was unaware of tools that have been introduced on the *TeX* market in recent times.

I really enjoyed reading each questionnaire as it turned out to be sort of a therapy session for me. “I’ve been there!” Some of the good-hearted humor was appreciated, for example

*Question:* “What sources of support did you use?”  
*Response:* “Sweat” and “Hours of Trial and Error”

or—

*Question:* “What do you think are *TeX*’s weak points?”

*Response:* “I won’t live long enough to master it.”

### **The Responses**

The percentages represented under a number of topics will not always add up to 100%. Several questions allowed for multiple responses, so the percentages represent a value in relation to the 60 respondents.

**Please note:** The “bulleted” items represent direct quotes taken from the questionnaire responses. Although some of the statements are inaccurate, the wording of the text taken from the responses has not been changed.

Regarding the content of the responses, please note that, although I have had many similar experiences, “the opinions expressed here do not necessarily reflect those of the management.” I have tried to give a sample of all types of responses so this will present both positive and negative aspects of working with *TeX* in a production environment.

The questions can be categorized roughly as follows:

1,2	Areas of interest and use
3,4	Hardware environment
5,7	Training, expertise
8	Macro packages
9,10	Fonts
11,14,15	Problems, weaknesses
12	Initial encounters
13	Strengths
16,17	Resources
18	Future involvement

Here goes ...

#### **1. What typeset product is the main output of your organization?**

The most popular type of page output being produced by the people responding to the questionnaire was for technical books and journals, taking advantage of *TeX*’s ability to produce high quality math.

Technical Books	46%
Journal	36%
Internal Documents	26%
Magazine	11%
General Topic Books	10%
Forms	10%
Directories	8%
Newspaper	6%
Labels	3%
Other	35%

Of those who responded “other,” the most common work was Training Materials and User Manuals as well as Technical and Software Manuals. In addition, the participants indicated using *TeX* to produce the following other types of output: Articles, Dictionaries, Documentation, Legal documents, Letters, Mathematics, Preprints, Proposals, Reports, Technical papers, and Theses and Dissertations.

## 2. Are you using **TEX** now for output of any typeset pages?

Ninety percent of those answering said that they are using **TEX** for at least a portion of their typeset pages; ten percent said they did not use it in production.

Of those who answered yes, the survey broke down to the following percentages of total pages produced in their environment:

under 25% of pages	16%
25 to 50% of pages	10%
50 to 75% of pages	8%
75 to 100% of pages	66%

Of those who said they are not using it in production now, 60% said they are experimenting with it for possible future use, and 40% said that they had decided to not use it in production.

## 3. In what environment are you using **TEX** — mainframe or micro?

About 34% of the users said that they were operating in a mainframe environment, 67% said that they were using some form of micro.

**Note:** Some respondents classified the SUN equipment as a mini or super-micro computer, others classified it as a mainframe. I have left the responses as is so SUN will appear in both breakdowns.

VAX (running VMS) was the most popular mainframe in use, with 25% of the survey, followed by 5% each for IBM, Hewlett-Packard, and SUN equipment. Other machines in use were: Amdahl OS/MVS/XA, DEC 2065 (TOPS-20), DG (AOS), NAS AS/9160, Pyramid 90X OSX, and VAX (VMS and UNIX).

In the micro-class machine, the largest share went to IBM PCs (XT, AT, 286, etc.) and clones (46%), followed by 15% using a Macintosh and 11% using SUN workstations. Other micros used by people in the survey were: Apollo DN 3000/4000/330, AT&T 6300, COMPAQ 286, DEC Unity 68 (UNIX), Cromemco CS420, 68020, IBM RT Workstation, Integrated Solutions 68010, Leading Edge Model D, Tandy 3000HL, Olivetti M24, Wyse PC286.

## 4. On what type(s) of device(s) are you producing output?

Many of the surveys indicated that **TEX** was being run on more than one type of output device. Quite often a laser printer was used for proofing and a typesetter was used for final camera copy, or several types of laser printers existed in their production environment.

Apple LaserWriters	38%
Cordata Corona	5%
DEC LN03	11%
HP Laser Printer	25%
IBM 3820 or Pageprinter	5%
Imagen 8/300 or other	13%
Talaris	8%
QMS (PS)	13%
Varietyper VT 600	18%

Other laser or impact printers indicated by individuals were:

AST Research/PostScript  
Canon LBP A1  
LN01  
QMS Kiss  
Panasonic Laser KX-P4450  
ScripTen

For those using typesetters, the most common equipment used was Linotron. The percentages for this and other typesetters represented in the survey are shown below:

APS micro-5 (Autologic)	6%
Compugraphic 8600	1%
Linotron	8%
Monotype Lasercomp	2%
VC570	1%
Varietyper 4300P	1%

Only 2 respondents were using outside service bureaus to produce pages. The bureaus used were: Stürzt AG (Würzburg) and ArborText.

## 4A. Is your proofing output produced on a different device than camera copy? If yes, have you had problems with font compatibility, and how have you solved them?

Those surveyed said that 46% were producing final pages on a different device than the one used for proofing. The majority (54%) said that the same machine was used for both proof and final copies.

When indicating problems that had arisen with sending the same file to two devices, the following types of comments were given:

### *General*

- No problems, we only use CM fonts
- Occasionally, but no problems
- Minor compatibility problems
- We're using Textures on the Mac
- Using another device for testing (with Monotype fonts)

#### *.TFM file problems*

- We had to hand generate .tfms to get proof output for our typesetter's fonts
- Renaming PS .tfms to CMR like .tfm files

#### *Previewers*

- Font incompatibility between the previewer and the typesetter
- Use Maxview to preview

#### *AM/CM Font complications*

- We have had to use AM fonts because APS doesn't have CM
- Fake font specs written to make AMR imitate final fonts

#### *Font generation*

- A lot of work to generate font files.
- Layout proof with fonts represented by boxes on a Tektronix compatible device

#### *Using PostScript*

- Use PostScript fonts for text, CM for math
- No major problems, using PostScript devices
- Metafont to PostScript software
- Our own .dvi to PS and CORA drivers, but have trouble with PI fonts

#### **5A. How do you feel about the level of training required to use *TeX* for typesetting? Please explain your answer.**

As compared to other typesetting systems, the vast majority of respondents (78%) indicated that they thought *TeX* required more training to get up to speed, 7% thought that it required less time, 9% rated it about the same, and 6% had no opinion because they had nothing to compare it to.

Representative of the types of comments on *TeX* training were the following:

#### *Those indicating "more" time*

- Intimidating for beginners
- Requires more, but pays off
- Lack of interactive and WYSIWYG
- If a user wishes to reuse an existing format the expertise is less, to create a new format the expertise is more.
- The users need to learn more about the internal workings of the tool than is true of some other tools, but can accomplish more.
- More than others but high quality output, substantial initial learning curve

- Meant for people who understand a markup language, not people used to simple screen editors.
- More, based on my experience with word-processing systems.
- More training but more capabilities
- You need a local *TeX*pert to get started, long learning curve
- High level of user needed
- Different level of need means different levels of training
- Hard to find answers in the *TeXbook*
- Long learning curve
- It provides more than word processing so it takes longer
- *TeXbook* is complicated so we had to develop our own training materials
- Technically oriented and beyond secretaries who must rely on macros
- Its not as user friendly as most commercial typesetting systems
- Macro writing is more like programming than typesetting
- Macro packages too restrictive for actual use
- Intense training required
- Requires some programming background

#### *Those indicating "less" time:*

- Commands are more English-words than most other mnemonic systems
- Less than most systems if you're doing math
- Find *TeX* easier to experiment with
- If the macros are set up well minimal training is involved

#### *Those who felt it was similar or the same:*

- Training time seems similar to other systems we have used
- We use TROFF and the learning curve seems to be the same
- Similar to WYSIWYG systems
- We use the same data input language as our old system
- Depends on how well the macros and style sheet are put together

#### *No opinion*

- Nothing to compare to
- Can't decide yet

For the sake of those that are recently getting involved with *TeX*, the last few years have seen the development of a number of introductions to *TeX* that make it easier to get started, for example:

- *First Grade T<sub>E</sub>X* by Arthur Samuel (available through the T<sub>E</sub>X Users Group)
- *A Gentle Introduction to T<sub>E</sub>X* by Michael Doob (available through the T<sub>E</sub>X Users Group or on many electronic bulletin boards)
- *Another Look at T<sub>E</sub>X* by Stephan Bechtolsheim (Due to be published in March of 1990, Springer-Verlag, also available in manuscript form from the author)
- A number of L<sub>A</sub>T<sub>E</sub>X books that have been produced in the last few years.

**6. Do your keyboarders really have to know T<sub>E</sub>X, or is it "hidden" from them? (Please explain.)**

The majority said that their keyboarders had to know T<sub>E</sub>X to produce their pages (54%), but nearly as many said that they kept the inner secrets of T<sub>E</sub>X from the production personnel (46%).

For those that indicated it was necessary to "know" T<sub>E</sub>X, the following comments are representative:

- They do now but we are working on a data entry system to limit this
- They don't have to but they like to
- They only have to know basic rules
- They like to know as much as possible
- They have to know T<sub>E</sub>X to debug errors
- Yes, they have to know it to format our files
- Have to know (used by software engineers, not secretaries)
- They know it to some degree, they don't code from scratch
- Only a very little for immediate needs
- For now, as others are added they will only be taught what they need
- They must know plain basics and A<sub>M</sub>S-T<sub>E</sub>X
- They understand the majority of the T<sub>E</sub>X functions

Those who keep T<sub>E</sub>X hidden gave the following remarks to clarify why:

- Some writers do, most know our macro package.
- Most are unfamiliar with plain
- We use A<sub>M</sub>S-T<sub>E</sub>X
- They know L<sub>A</sub>T<sub>E</sub>X macros
- We teach them only our macro names
- Only technical people know

**7. Who creates the code for output routines, etc., in your environment? (Explain)**

The most common response was that an in-house guru (T<sub>E</sub>Xpert) was needed to keep T<sub>E</sub>X running

smoothly (60%). Some started with consultants and switched over to in-house support (12%). Others have all style files done by their production personnel (18%) and the remainder purchase packages or avoid changing too many things (10%).

*In-house T<sub>E</sub>Xpert*

- Used to change style sheets
- In-house experts adjust style files and fonts
- Done by our programmers

*Consultant*

- Started with a consultant but now doing it in-house
- ArborText wrote our original macros

*Production personnel*

- We use only L<sub>A</sub>T<sub>E</sub>X with minor adjustments

*Other*

- In-house macros and L<sub>A</sub>T<sub>E</sub>X
- Barb Beeton did most of the work originally for the output routines
- We don't use custom output routines
- Purchased package
- PCT<sub>E</sub>X package

**8. Do you use Plain T<sub>E</sub>X or a "standard" macro package? Which package(s)?**

The vast majority of users (71%) indicate a preference for plain T<sub>E</sub>X for, as one user put it, "it's sheer power". The next most popular package was L<sub>A</sub>T<sub>E</sub>X (40%). A smaller number use A<sub>M</sub>S-T<sub>E</sub>X (13%) and 16% indicated another package. Of the "others", 75% indicated that they had to develop their own in-house macro package to meet their production needs.

**9. Where and how do you get fonts not delivered with the standard T<sub>E</sub>X release?**

Almost all the respondents indicated some use of the "standard" fonts distributed with T<sub>E</sub>X.

Sixteen percent indicated that they used only the standard fonts.

In the "Beg, Borrow, and Steal" category, 6% indicated that they either "scanned and produced PK files with our own software, and created some special fonts ourselves" or got them from "various archive sites via network".

The majority (80%) indicated that it was necessary to go to other font sources to meet their production needs. The majority use one of the following 4 sources:

ArborText .tfms	
with typesetter's fonts	8%
Bitstream	16%
Talaris Systems, Inc.	8%

Other small percentages indicated other font sources:

AMS fonts	
Autologic TR fonts from T <sub>E</sub> XSource	
Autologic fonts using ArborText software	
Berkeley font library	
Compugraphic (.tfms designed in-house)	
Danish Linotype agent	
From DECUS for Digital LN03	
Folio	
Using FTP	
Metafoundry	
PostScript fonts and METAFONT	
University of Manitoba	

#### 10. Have you used METAFONT at all in your installation? Explain.

The majority (55%) indicated that they had not used METAFONT; the rest (45%) said they had, but most of them had used it only for small applications.

##### *Comments from those who have not used METAFONT*

- Haven't had the time
- No we're not typeface designers
- Installed but untried
- Received but not working yet

##### *Comments from those who used METAFONT*

- Only with standard METAFONT files
- To build simulation fonts
- To make logos
- To develop special math and foreign language symbols
- Experimental only
- Not to a great extent
- To initialize fonts
- To make some new mag steps for fonts
- Translate fonts from old MF format to new
- To develop new fonts
- Tuning fonts

#### 11. What have been some of the problems you have encountered trying to develop the use of T<sub>E</sub>X in your environment?

There was a variety of responses here. They ranged from taking too much time to train personnel to frustrations about trying to get support (even if they were willing to pay for it). I have tried to

group the comments to similar problems and show a representative sampling of the comments submitted.

##### *ASCII vs. EBCDIC — 3%*

- IBM mainframe not suited for T<sub>E</sub>X data entry
- We are EBCDIC oriented and have to learn set up for ASCII

##### *Documentation — 11%*

- No Documentation for beginners
- Difficulty in understanding *The T<sub>E</sub>Xbook*
- Hard to look up answers to problems in the *T<sub>E</sub>Xbook* (you have to look in 3 or 4 places to find out how one command works).
- *The T<sub>E</sub>Xbook* does not explain the interaction between basic commands and you have to experiment to find out what will happen

##### *Error Messages / Debugging — 3%*

- Error messages are useless to a novice
- We have not found a source for many error messages encountered

##### *Fonts — 18%*

- Implementing Scandinavian hyphenation patterns
- Lack of compatible fonts for our typesetters
- Font development and maintenance
- Translate fonts from old METAFONT format to new
- Getting some of the Bitstream Fonts to work with CM fonts

##### *Foreign languages — 3%*

- Getting foreign language characters
- Foreign language hyphenation

##### *Getting users to accept system — 6%*

- Hostility from users of a previously used system
- Users don't appreciate the quality
- Most casual users don't appreciate the quality difference so they don't want to put in the time to learn

##### *Graphics — 3%*

- Integration of graphics with output

##### *Macros / Output routines — 23%*

- Updating macros
- Incompatibility with other DOS applications
- Time consuming to write macros for T<sub>E</sub>X
- New formats are a struggle
- Output routines are a misery to debug
- Output routines are the hardest
- Multi-column output and page balancing

- Macros written before I started here, hard to change

*Output devices/drivers — 10%*

- Getting the right output devices, good output devices
- VAX C bugs while developing a double sided DVITOLN3 program.
- Rounding errors on device drivers
- Problems with some current .dvi drivers

*Production problems — 6%*

- Slowness of proofing documents
- Implementing changes without disrupting publications in progress
- Preprocessor needed to facilitate typing
- Setting narrow columns

*Support — 6%*

- Unable to purchase *TEX* with support contracts for our system.
- Cannot purchase a service contract even if you are willing to pay for it.
- Assistance with problems
- Support during set up

*System initialization — 3%*

- Understanding *TEX*, C*TEX*, etc.
- Trying to decide between *TEX* and L*A**T**E**X*

*System requirements — 3%*

- CPU intensity takes too many computer dollars for a lot of users
- Running out of *TEX* memory

*TEX algorithms and design — 10%*

- Page Breaking problems
- Runarounds (parshapes)
- Problems with inserts

*Training time/Learning curve — 20%*

- Takes too much time for busy people to learn it
- Initial learning curve/training
- Hard to teach to people who don't have typesetting background

**12. How did you find out about *TEX*?**

Responses here were varied. The largest group (19%) indicated that they had found out about *TEX* through their jobs (several said "I inherited *TEX* from my predecessor"). The next largest percentage (17%) indicated that they had found out from the source, Knuth at Stanford (and his papers) or

The *TEXbook* and *The METAFONTbook*; an equal number from contacts at a university. Word of mouth from other users was the source for 13%. Five percent or less found out from a consultant, the physics community, or AMS and *Mathematical Reviews*.

Other minor sources were as listed: Decus, Friend at the Federal Reserve, McGraw-Hill recommended it, Scientific/Technical Institutes, reading, tried to turn it into a product, from customers, Trade Magazine (*Mac User*), reading about the SAIL version, through a typesetter, and classes in *TEX*.

**13. What do you feel are *TEX*'s strong points?**

By a clear margin, users indicated *TEX*'s design (81%) and flexibility (53%) as the strong points of the *TEX* language. Portability is also important in production environments. If the language can be moved to another system, there is no need for re-training "because the old composition system doesn't work on the new mainframe." A sampling of comments on these and other features are:

*Batch orientation — 2%*

- Batch oriented so it can be linked with pre- and post-processors

*Design points — 81%*

- Conditionals
- Control over look of output
- Dynamic control of vertical spacing
- Ease of Macro construction
- Capabilities you can get with macros
- File handling, ability to import files
- Error reporting
- High quality output/Typographic quality
- Hyphenation algorithms
- Line-breaking and hyphenation algorithms
- Nice displays and easy setting of page composition
- Page bottoming
- Powerful in designing format but painful to write it
- Widow control
- Ability to build indices and glossaries
- The ability to use the same file to create different page shapes
- It understands many typographers conventions which other systems have to be taught
- Tables
- Kerning
- Richness of the language

- Its essentially a programming language, you get lots of power with it
- Calculation capabilities
- Ability to handle low-level formatting
- Paragraph building algorithm
- Automatic pagination
- Speed

*Flexibility* — 53%

- Ability to program what you want
- Flexibility, not limited like other packages
- Precision and reliability
- Programmable
- Primitiveness, which allows control

*Mathematics* — 25%

- Mathematics/equation typesetting
- Sophisticated mathematics

*Portability* — 13%

- Portability
- Availability on PCs
- Device independent output
- Portability of TeX documents if you limit your macro use
- The possibility of linking other programs to TeX
- System independence

*Price* — 5%

- Affordable
- Public domain

**14. What do you feel are TeX's weak points?**

The major complaint of the respondents was the lengthy "learning curve" and training time involved to get TeX up and producing pages in a profitable manner (37%). Next were TeX design points that had caused production problems (35%). The next highest indicated problem was the lack of standard graphics support (18%) as part of TeX's design.

In addition, they submitted comments on these and other areas that they felt were weak points of TeX.

*Batch* — 10%

- Batch process
- Runaway errors in batch process
- Lack of interaction

*Documentation* — 5%

- Hard to find novice documentation
- Lack of a complete reference document
- Lack of beginning level documentation

*Errors/Debugging* — 5%

- Lack of error diagnostics, hard to understand error messages
- Lack of decent debugging support for macro programming

*Fonts* — 5%

- Fonts/font management
- Changing font sizes

*Foreign language support* — 5%

- Foreign languages, fonts and hyphenation

*Graphics* — 18%

- Chemistry
- Use of marks
- Lack of standard graphics handling
- Graphics (figure) support is poor

*Macro files/Output routines* — 13%

- A lot of set up work to create style files
- Non-trivial layout is almost impossible
- Difficulty of setting multi-column output
- Hyphenation and overfull boxes
- Inserts of more than 1 column

*Previewing/WYSIWYG* — 8%

- No immediate previewing
- No WYSIWYG

*Production problems* — 5%

- Landscape tables
- Hard to write macros
- Not user friendly

*Support* — 8%

- Not supported by major computer vendors
- Need of a guru for support

*System considerations* — 8%

- Requires a lot of computer resource
- Slow in PASCAL
- Running out of memory
- Uses a lot of room
- Unavailability of front-end processors

*Training/Learning curve* — 36%

- Not great for beginners
- Hard to learn
- Amount of knowledge it requires
- Its not that you can't do things, it's that it is very time consuming to figure out how
- Not easy to use, especially if you're in a hurry
- Not easy to learn

- Extremely complex, I won't live long enough to master it
- Complexity
- Documentation
- Finding information from the TeXBook can be difficult
- Lengthy learning curve
- You really have to know what you are doing, its for gurus
- Hard to learn without help

*TeX design — 40%*

- Pagebreaking algorithms
- Lack of totally integrated system
- Many modes and their idiosyncrasies
- No `\everyline` command
- Primitiveness which requires complete specifications for everything
- Setting Tables
- Inability to control letterspacing and kerning in a global environment
- You can never be confident that even proven techniques and macros will work like you think
- Not enough idioms, macro packages tend to be limiting without intimate understanding
- Pagination/Page breaking (ignores my `\goodbreaks`)
- Adjusting page breaks in a long document.
- Handling final pagination of output
- No hooks to other languages or system commands
- Difficult user interface

**15. What would you change about TeX if you could?**

Of course hindsight is always 20/20, but some suggestions were realistic in the aid that they could provide to users with large page output needs.

A small percentage of people said "nothing" or "too soon to say," but most respondents were quite vocal about what they would change in the design of TeX if they could. Many of the problems listed here have been taken care of by products or macro packages written in the last calendar year so we can see progress being made towards smoother production control.

*Batch processing vs. Interactive — 6%*

- Better batch processing
- Interactive paging
- Provide a page by page operation (set, correct, move to next page)

*Documentation — 10%*

- User friendly manual

- Better or more manuals
- Better documentation
- More intermediate documentation needed
- Write a comprehensive guide organized by command
- Make a separate tutorial for tables and equations

*Errors and debugging — 6%*

- Make it more user friendly and easy to debug from error messages.
- It would be nice if a programmer could gain access to over- and under-full box information during processing
- True debugging capabilities

*Fonts — 5%*

- Make it easier to change fonts and sizes
- If you could use all 256 characters in a standard font
- Font management

*Foreign languages — 2%*

- Add multi-language supports

*Graphics — 10%*

- Better standardized graphics support
- Include PostScript graphics as standard
- Include chemical structure manipulation

*Installation/System setup — 4%*

- Add more READ.ME files with distribution to help installation

*Macros/Output routines — 6%*

- More macro packages
- Set up of output routines
- Flexible macro package with a collection of well documented parameters
- More standardization of macros so that several packages can work together

*Previewing — 10%*

- More of a WYSIWYG tool
- Make preview interactive
- Preview of non CM fonts

*System requirements — 2%*

- Change the hard disk requirements for its fonts

*Training — 8%*

- Cheaper courses
- Once you start to do anything complex you are basically programming

*TEX design* — 25%

- Add a rotate box
- The ability to run a single page in a large document
- The definition of *sp* to a larger value
- More “mark” capabilities
- Have the concept of a “spread” in *TEX*
- Would like to be able to mask output selectively (for color)
- Add the ability to delay execution of commands until later pages
- Consistent syntax
- Make verbatim environments easier
- Make arbitrary placement of text on the page easier.
- Would be nice to be able to turn off *TEX*’s paragraph composition mechanisms and use a line-by-line approach when needed.
- Palettes for esoteric math symbols
- Give user more control of line and page-breaking if needed
- Part of .dvi values (x,y positions) should be accessible in *TEX*
- A command like `\unhbox` should give the text and not just char boxes

*TEX and other languages* — 3%

- Hooks to other languages
- Pagebreaking algorithms

*User interface* — 10%

- Make the help function more helpful
- Mouse driven interface
- Write conversion programs for popular word-processing packages
- Previewers needed for more devices
- Add a pre-processor
- Would be nice to have a version of *TEX* which did not expect user interaction, not really suited for high-volume work

**16. What sources have you used to help with *TEX* problems? (Please explain)**

The vast majority (88%) used *The TEXbook* as at least one of their sources for support, followed by 65% of users who get information from *TUGboat*. The rest of the list is shown below:

Knuth’s <i>The TEXbook</i>	88%
Copies of <i>TUGboat</i>	65%
Courses offered in <i>TEX</i>	31%
<i>TEXhax</i>	28%
<i>TEX Users Group</i>	16%
AMS office	8%
<i>TEXmag</i>	8%

Local <i>TEX</i> group meetings	5%
Other	38%

Those indicating “other” sources of problem-solving had these remarks:

- In-house courses
- Experimentation (Hours of)
- Sweat!
- Looking at macros other people had written
- other experienced users with similar interests
- Network news group
- Friends who know *TEX*
- UK*TEX*
- *TEXline*
- S. Bechtolsheim’s *Another Look at TEX*
- *TEX* Users Group Advanced *TEX* and Macro Writing courses
- *TEX* Users Group used to be good for phone support but now directs me to other local *TEX* Users Group members for help
- Addison-Wesley
- Arbortext
- *LATEX* book
- Personal *TEX*
- Mike Spivak
- *Joy of TEX*
- German books from *TEX* gurus

**16A. If you have contacted the *TEX* Users Group, were they able to answer your question or solve your *TEX* problem for you?**

Sixty percent of the callers were able to get help from the *TEX* Users Group headquarters, 40 percent were not.

Of those who have contacted the *TEX* Users Group, they say:

- Helpful, especially since they hired the support person
- Barbara Beeton is extremely helpful
- Missed an issue of *TUGboat*
- Signed up for *TEX* Users Group courses

Those who did not get help responded:

- Have never contacted, do they do debugging?
- When I called I was told they were working on acquiring support staff
- Asked for info if it ran on a 386 micro, they didn’t know
- They didn’t have the information but it later appeared in *TUGboat*
- Asked a question and they directed me to a consultant

**17. Can you think of any areas where the TeX Users Group could be of help to you?**

Although some surveys indicated that the TeX Users Group is "doing a great job now" they had suggestions for future improvements or ways that TUG could use their influence to help make their TeX lives easier.

*Fonts*

- Commissioning sources for new fonts
- Help with font compatibility

*Local Support*

- Start a local chapter in New Jersey

I and a gentleman by the name of Bob Jantzen of Villanova University started our own local TeX users group (which we call the Delaware Valley TeX User's Group). We started by contacting Ray and Karen to get a copy of the mailing list for all registered users in our zip-code territory. We started with an original mailing, and have updated the list based on interest in our area. We have meetings 6 times a year, and take turns making presentations of different macros and output routines that we have developed. We usually share paper or soft copies.

*Macros*

- Push for standardization of macro packages.
- Development of macro packages

*Product information*

- New products developed in TeX environment
- Keep an updated database of TeX products and sources
- More addresses for TeX products

*TeX design*

- Push for improvements in TeX where needed

*TeX PR*

- More PR to heighten awareness of TeX, perhaps an article in widely read engineering magazines

*Training*

The price of training was out of reach for many smaller operations by the time you took into account airfare, hotels, meals and tuition. Some people suggested the development of a beginning level correspondence course that the user could walk through and maybe contact TUG when problems arose.

- Development of a beginning level correspondence course for poorer TeX users
- Reduce the price on their course offerings
- Some low-cost training
- Sponsoring scholarships to courses
- Sponsoring the writing of more TeX support books
- Maybe some videotape training on TeX and LATEX
- More classes
- Provide more documentation
- Would like to see a practical "how to" User's guide to PLAIN TeX published (the TeXbook is more academically oriented)

*TUGboat*

A number of users felt that the articles in *TUGboat* were beyond their level of comprehension and were looking for more support in less tricky solutions to everyday publishing problems. Several suggested including a few beginner's articles each month.

- More information on what other commercial publishers are doing
- Issue *TUGboat* more frequently
- Circulate more information and advice to novice/intermediate users
- Could cater to the mid-range TeX users more, most *TUGboat* articles are too advanced for me.
- Focus more articles in *TUGboat* on real typesetting problems rather than esoteric concepts
- More articles for beginners

*Support*

People who have deadlines are frustrated when they are trying to figure out some of TeX's more obscure bugs when creating macros or using primitives for the first time. They are willing to pay for support, but often cannot easily get it.

- More technical people at site to answer questions
- Make TeX Users Group members aware of where they can get technical support
- E-mail address for asking questions and getting responses
- Have phone support available for hard to find bugs
- Organize a periodic printout of TeXhax for those of us not on a network
- Give me support and charge
- TeX Users Group notifying users by electronic mail of TeX source updates
- Provide sources for updating TeX '82

**18. What do you anticipate will be your future involvement with  $\text{\TeX}$ ?**

The majority of surveys (74%) indicated that they would be continuing use of  $\text{\TeX}$  for some or all of their typesetting output. About 18% indicated that they felt their use would decline in the upcoming months for a variety of reasons. Eight percent felt that their involvement would be in developing tools to use with  $\text{\TeX}$ .

*Continuing with  $\text{\TeX}$  typesetting*

- More involvement in formatting documents, style sheets.
- Ongoing use of  $\text{\TeX}$  on Novell Network of PCs
- More use of  $\text{\TeX}$  probably with PostScript
- More active because of use by Physics community
- Working with *The Publisher* from ArborText
- Typesetting college level texts
- More use of  $\text{\TeX}$  because of page preview capabilities
- Continued use at work and at home
- Develop production standards for our courses
- Continued work with it at my job
- Would like to attend annual meeting and Wizard class
- Local  $\text{\TeX}$  guru, member of  $\text{\TeX}$  Users Group
- Continue production of company documents with  $\text{\TeX}$
- I'll keep using it, cursing periodically.
- Lots of multi-column work
- Continued use for Journal work
- Continued production of books
- It handles math so well we will continue to use it
- We will continue to use it and refine our own macro package
- Work my way to  $\text{\TeX}$ pert
- Expanded use for technical books and articles
- Graphics user interface
- Investigate using other fonts through a PostScript driver to produce engineering books
- More, hope to also learn METAFONT
- Intend to use it as our primary publishing vehicle indefinitely.
- Introducing more fonts to use with  $\text{\TeX}$
- Offering total manuscript production via  $\text{\TeX}$
- Continued involvement
- Continued involvement for math work

*Declining use*

- Looking for a completely integrated documentation package

- We will probably be replacing with other tools such as Ventura Publisher
- Unfortunately when I leave here,  $\text{\TeX}$  will be replaced with a word-processing package
- Unless future versions of  $\text{\TeX}$  are screen oriented and "friendly" we will go to scientific word processors as they become available.
- If I can predict, I would not be doing  $\text{\TeX}$  in the future
- We expect to be typesetting our entire newspaper in the future, but expect it to disappear when we go to electronic full page makeup
- Uncertain
- Will use for some things, but are looking for another system for some publications

*Driver and other tool development*

- Continue to use it but look for better integrated environments to run  $\text{\TeX}$
- Developing new printer drivers
- Parsing SGML to  $\text{\TeX}$
- Working on a better previewer for us

**Some Conclusions**

This survey produced a lot of useful information that could help in sparking some ideas and activity from people involved with  $\text{\TeX}$  from a number of different levels. There are some ideas for the  $\text{\TeX}$  Users Group to use in their future support of the  $\text{\TeX}$  community. For the advertisers, make your products better known, or start to develop in areas that are of concern to production typesetters. To  $\text{\TeX}$  support personnel, let us know where you are, and be willing to provide service and support (for a fee) to people who are driven by tough production deadlines.

If anyone has any other ideas that would be for the common good, maybe we could start a  $\text{\TeX}$  "Publisher's Corner" providing tips in future copies of the *TUGboat*. Any volunteers to get us started?

◊ Elizabeth M. Barnhart  
National EDP Department  
TV Guide  
#4 Radnor Corporate Center  
Radnor, PA 19088

## Software

### Exercises for *T<sub>E</sub>X: The Program*

Donald E. Knuth

During the spring of 1987 I taught a course for which Volume B of *Computers & Typesetting* was the textbook. Since that book was meant to serve primarily as a reference, not as a text, I needed to supplement it with homework exercises and exam problems.

The problems turned out to rather interesting, and they might be useful for self-study if anybody wants to learn *T<sub>E</sub>X: The Program* without taking a college course. Therefore I've collected them here and given what I think are the correct answers.

The final problem, which deals with the typesetting of languages that have large character sets, is especially noteworthy since it presents an extension of *T<sub>E</sub>X* that might prove to be useful in Asia.

Some of the problems suggested changes in the text. I've changed my original wording of the problem statements so that they will make sense when the next printing of the book comes out; people who have the first edition should check the published errata before looking too closely at the questions below. But some problems (e.g. 25, 26, and 32) assume the old 7-bit version of *T<sub>E</sub>X*.

**Editor's note:** The answers to the following exercises will appear in the November 1990 issue of *TUGboat*.

### The Problems

Here, then, are the exercises in the order I gave them. Although they begin with a rather "gentle introduction," I recommend that the first ones not be skipped, even if they may appear too easy; there often is a slightly subtle point involved. Conversely, some of the problems are real stumpers, but they are intended to teach important lessons. A serious attempt should be made to solve each one before turning to the answer, if the maximum benefit is to be achieved.

1. (An exercise about reading a WEB.) In the Pascal program defined by the book, what immediately precedes 'PROCEDURE INITIALIZE'? (Of course it's a semicolon, but you should also figure out a few things that occur immediately before that semicolon.)

2. Find an unnecessary macro in §15.
3. Suppose that you want to make *T<sub>E</sub>X* work in an environment where the input file can contain two-character sequences of the form '*esc* *x*', where *esc* is ASCII character number '33' and where *x* is an ASCII character between '0' and '-' inclusive. The result should be essentially equivalent to what would have happened if the single (possibly nonprinting) ASCII character *chr*(*ord*(*x*) - '100') had been input instead. If *esc* appears without being followed by an *x* in the desired range, you should treat it as if the *esc* were ASCII character number '177'.

For example, the line 'A *esc* A *esc* a eoln' should put four codes into the buffer: '101, '001, '177, '141.

What system-dependent changes will handle this interface requirement?

4. Suppose that the string at the beginning of the *print\_roman.int* procedure were "m2d5c212q5v5i" instead of "m2d5c215x2v5i". What would be printed from the input 69? From the input 9999?
5. Why does *error\_count* have a lower bound of -1?
6. What is printed on the user's terminal after 'q' is typed in response to an error prompt? Why?
7. Give examples of how *T<sub>E</sub>X* might fail in the following circumstances:
  - a) If the test '*t* ≤ 7230584' were eliminated from §108.
  - b) If the test '*s* ≥ 1663497' were eliminated from §108.
  - c) If the test '*r* > *p* + 1' were changed to '*r* > *p*' in §127.
  - d) If the test '*rlink*(*p*) ≠ *p*' were eliminated from §127.
  - e) If the test '*lo\_mem\_max* + 2 ≤ *mem\_bot* + *max\_halfword*' were eliminated from §125.
8. The purpose of this problem is to figure out what data in *mem* could have generated the following output of *show\_node\_list*:

```

\hbox(10.0+0.0)x100.0, glue set 10.0fill      100
.\discretionary replacing 1                   200
..\kern 10.0                                300
.|\large U                                    10000
.|\large ^~K (ligature ff)                  400,10001,10002
.\large !                                     10003
.\penalty 5000                               500
.\glue 0.0 plus 1.0fill                     600
.\vbox(5.0x0.5)x10.0, shifted -5.0       700
..\hbox(5.0x0.0)x10.0                      800
... \small d                                 10004
... \small a                                 10005
.. \rule(0.5+0.0)x*                         900

```

Assume that `\large` is font number 1 and that `\small` is font number 2. Also assume that the nodes used in the lower (variable-size) part of *mem* start in locations 100, 200, etc., as shown; the nodes used in the upper (one-word) part of *mem* should appear in locations 10000, 10001, etc. Make a diagram that illustrates the exact numeric contents of every relevant *mem* word.

**9.** What will *short\_display* print, when given the horizontal list inside the larger `\hbox` in the previous problem, assuming that *font\_in\_short\_display* is initially zero?

**10.** Suppose the following commands are executed immediately after TeX has initialized itself:

```

incr(prev_depth);
decr(mode_line);
incr(prev_graf);
show_activities.

```

What will be shown?

**11.** What will '`show_eqtb(int_base + 17)`' show, after TeX has initialized itself?

**12.** Suppose TeX has been given the following definitions:

```

\def\af{\advance\day by 1\relax}
\def\gf{\global\af}

```

The effect of this inside TeX will be that an appearance of `\af` calls

`eq_word_define(p, eqtb[p].int + 1)`,

and an appearance of `\gf` calls

`geq_word_define(p, eqtb[p].int + 1)`

where  $p = int\_base + day\_code$ . Consider now the following commands:

```
\day=0 \g\af\af\gf\af\gf\af\af\af\af
```

Each '{' calls *new\_save\_level(simple\_group)*, and each '}' calls *unsafe*.

Explain what gets pushed onto and popped off of the *save\_stack*, and what gets stored in *eqtb[p]* and *xeq\_level[p]*, as the above commands are executed. What is the final value of `\day`? (See *The TeXbook*, exercise 15.9 and page 301.)

**13.** Use the notation at the bottom of page 122 in *TeX: The Program* to describe the contents of the token list corresponding to `\!` after the definition

```
\def\!!1#2![{!#}#!]2}
```

has been given, assuming that [, ], and ! have the respective catcodes 1, 2, and 6, just as {, }, and # do. (See exercise 20.7 in *The TeXbook*.)

**14.** What is the absolute maximum number of characters that will printed by `show_eqtb(every_par_loc)`, if the current value of `\everypar` does not contain any control sequences? (Hint: The answer exceeds 40. You may wish to verify this by running TeX, defining an appropriate worst-case example, and saying

```
\tracingrestores=1
\tracingonline=1
{\everypar{}}
```

since this will invoke `show_eqtb` when `\everypar` is restored.)

**15.** What does INITEX do with the following input line? (Look closely.)

```
\catcode``=7 `` ``(``)``!
```

**16.** Explain the error message you get if you say

```
\endlinechar='! \error
```

in plain TeX.

- 17.** Fill in the missing macro definition so that the program

```
\catcode`?=active
\def\answer{...}
\answer
```

will produce precisely the following error message when run with plain *TEX*:

```
? Undefined control sequence.
<recently read> How did this happen?
```

### 1.3 \answer

?

(This problem is much harder than the others above, but there are at least three ways to solve it!)

- 18.** Consider what *TEX* will do when it processes the following text:

```
{\def\t{\gdef\aa{\catcode`d=12\tid#2#3{#2}}}
\hfuzz=100P\ifdim12pt=1P\expandafter\aa
\expandafter\else\romannumeral888\relax\fi
\showthe\hfuzz \showlists
```

(Assume that the category codes of plain *TEX* are being used.)

Determine when the subroutines *scan\_keyword*, *scan\_int*, and *scan\_dimen* are called as this text is being read, and explain in general terms what results those subroutines produce.

- 19.** What is the difference in interpretation, if any, between the following two *TEX* commands?

```
\thickmuskip=-\thickmuskip
\thickmuskip=-\the\thickmuskip
```

(Assume that plain *TEX* is being used.) Explain why there is or isn't a difference.

- 20.** In what way would *TEX*'s behavior change if the assignment at the end of §508 were changed to '*b* ← (*p* = null)'?

- 21.** The initial implementation of *TEX82* had a much simpler procedure in place of the one now in §601:

```
procedure dvi_pop;
begin if dvi_ptr > 0 then
  if dvi_buf[dvi_ptr - 1] = push then decr(dvi_ptr)
  else dvi_out(push)
else dvi_out(push);
end;
```

(No parameter *l* was necessary.) Why did the author hang his head in shame one day and change it to the form it now has?

- 22.** Assign subscripts *d*, *y*, and *z* to the sequence of integers

2718281828459045

using the procedure sketched in §604. (This is easy.)

- 23.** Find a short *TEX* program that will cause the *print\_mode* subroutine to print 'no mode'. (Do not assume that the category codes or macros of plain *TEX* have been preloaded.) Extra credit will be given to the person who has the shortest program, i.e., the fewest tokens, among all correct solutions submitted.

- 24.** The textbook says in §78 that *error* might be called within *error* within a call of *error*, but the recursion cannot go any deeper than this.

Construct a scenario in which *error* is entered three times before it has been completed.

- 25.** (The following question was the main problem on the midterm exam.) Suppose WEB's conventions have been changed so that strings are not identified by their number but rather by their starting position in the *str\_pool* array. The *str\_start* array is therefore eliminated.

Strings of length 1 are still represented by their ASCII code values; all other strings have values ≥ 128, and they appear in the *pool\_file* just as before, in increasing order of starting position. The special code '128' is assumed to terminate each string.

Thus, for example, if such a WEB program uses just the three strings "ab", "", and "cd" (in this order), they will be represented in the corresponding Pascal code by the respective integers 128, 131, and 132. The program in this case is expected to initialize *str\_pool* locations 128–134 to the successive code values 97, 98, 128, 128, 99, 100, 128.

Such a modification requires lots of changes to *TEX*. Your job in this problem is to indicate what those changes should be. However, you needn't specify a complete change file; just say how you would modify §38–§48, §59, §259, §407, §464, and §602 (if these sections need to change at all). The other places where *str\_start* appears can be changed in similar ways, and you needn't deal with those.

Some of the specified sections will require new code; you should supply that code. Other sections may change only a little bit or not at all; you should just give the grader sufficient explanation of what should happen there.

**26.** Continuing problem 25, discuss briefly whether or not it would be preferable (a) to store the length of each string just before the first character, instead of using '128' just after the last character; or (b) to eliminate the extra '128' entirely and to save space by adding 128 to the final character.

**27.** J. H. Quick (a student) thought he spotted a bug in §671 and he was all set to collect \$40.96 because of programs like this:

```
\vbox{\moveright 1pt\hbox to 2pt{}  
 \xleaders\lastbox\vskip 3pt}
```

**28.** When your instructor made up this problem, he said '\hbadness=-1' so that TeX would print out the way each line of this paragraph was broken. (He sometimes wants to check line breaks without looking at actual output, when he's using a terminal that has no display capabilities.) It turned out that TeX typed this:

Loose \hbox (badness 0) in paragraph at lines 11--16  
[]\tenrm When your instructor made up this problem, he said

Tight \hbox (badness 3) in paragraph at lines 11--16  
\tenrm '\tentt \hbadness=-1\tenrm' so that T[]X would print out the way each

Tight \hbox (badness 20) in paragraph at lines 11--16  
\tenrm line of this paragraph was broken. (He sometimes wants to

Loose \hbox (badness 1) in paragraph at lines 11--16  
\tenrm check line breaks without looking at actual output, when

Loose \hbox (badness 1) in paragraph at lines 11--16  
\tenrm he's using a terminal that has no display capabilities.) It

Why wasn't anything shown for the last line of the paragraph?

**29.** How would the output of TeX look different if the *rebox* procedure were changed by deleting the statement '*if type(b) = vlist\_node then b ← hpack(b, natural)*'? How would the output look different if the next conditional statement, '*if (is\_char\_node(p)) ...*' were deleted? (Note that box *b* might have been formed by *char\_box*.)

**30.** What spacing does TeX insert between the characters when it typesets the formulas *\$x==1\$, \$x++1\$, and \$x,,1\$*? Find the places in the program where these spacing decisions are made.

**31.** When your instructor made up this problem, he said '\tracingparagraphs=1' so that his transcript file would explain why TeX has broken the paragraph into lines in a particular way. He also said '\pretolerance=-1' so that hyphenation would be tried immediately. The output is shown on the next page; use it to determine what line breaks would

(He noticed that TeX would give this vbox a width of 2 pt, and he thought that the correct width was 3 pt.) However, when he typed \showlists he saw that the leaders were simply

```
\xleaders 3.0  
.hbox(0.0+0.0)x2.0
```

and he noticed with regret the statement

```
shift_amount(cur_box) ← 0
```

in §1081.

Explain how §671 would have to be corrected, if the *shift\_amount* of a leader box could be nonzero.

have been found by a simpler algorithm that breaks one line at a time. (The simpler algorithm finds the breakpoint that yields fewest demerits on the first line, then chooses it and starts over again.)

**32.** Play through the algorithms in parts 42 and 43, to figure out the contents of *trie\_op*, *trie\_char*, *trie\_link*, *hyf\_distance*, *hyf\_num*, and *hyf\_next* after the statement

```
\patterns{a1bc 2bcd3 ab1cd}
```

has been processed. Then execute the algorithm of §923, to see how TeX uses this efficient trie structure to set the values of *hyf* when the word *aabcd* is hyphenated. [The value of *hn* will be 5, and the values of *hc[1..5]* will be (96, 96, 97, 98, 99), respectively, when §923 begins.]

```
% This is the paragraph-trace output referred to in Problem 31:  
[]\tenrm When your in-struc-tor made up this prob-lém, he  
@ via @@0 b=0 p=0 d=100  
@@1: line 1.2 t=100 -> @@0  
said '\tentt \tracingparagraphs=1\tenrm ' so that his tran-script  
@ via @@1 b=4 p=0 d=196  
@@2: line 2.2 t=296 -> @@1  
file would ex-plain why T[]X has bro-ken the para-  
@ \discretionary via @@2 b=175 p=50 d=46725  
@@3: line 3.0- t=47021 -> @@2  
graph  
@ via @@2 b=25 p=0 d=1225  
@@4: line 3.3 t=1521 -> @@2  
into lines in a par-tic-u-lar way. He also said  
@ via @@3 b=69 p=0 d=6241  
@@5: line 4.1 t=53262 -> @@3  
'\tentt \pretolerance=-1\tenrm ' so that hy-phen-ation would be  
@ via @@5 b=43 p=0 d=2809  
@@6: line 5.1 t=56071 -> @@5  
tried im-me-di-ately. The out-put is shown on the next  
@ via @@6 b=0 p=0 d=100  
@@7: line 6.2 t=56171 -> @@6  
page; use it to de-ter-mine what line breaks would  
@ via @@7 b=153 p=0 d=36569  
@@8: line 7.0 t=92740 -> @@7  
have  
@ via @@7 b=34 p=0 d=1936  
@@9: line 7.3 t=58107 -> @@7  
been found by a sim-pler al-go-rithm that breaks  
@ via @@8 b=1 p=0 d=10121  
@@10: line 8.2 t=102861 -> @@8  
one  
@ via @@9 b=15 p=0 d=10625  
@@11: line 8.1 t=68732 -> @@9  
line at a time. (The sim-pler al-go-rithm finds  
@ via @@10 b=164 p=0 d=40276  
@@12: line 9.0 t=143137 -> @@10  
the  
@ via @@10 b=0 p=0 d=100  
@ via @@11 b=192 p=0 d=40804  
@@13: line 9.0 t=109536 -> @@11  
@@14: line 9.2 t=102961 -> @@10  
break-point that yields fewest de-mer-its on the  
@ via @@12 b=174 p=0 d=33856  
@@15: line 10.0 t=176993 -> @@12  
first  
@ via @@12 b=41 p=0 d=12601  
@ via @@13 b=75 p=0 d=7225  
@ via @@14 b=75 p=0 d=7225  
@@16: line 10.1 t=110186 -> @@14  
line, then chooses it and starts over again.)  
@\\par via @@15 b=0 p=-10000 d=10100  
@\\par via @@16 b=0 p=-10000 d=100  
@@17: line 11.2- t=110286 -> @@16
```

**33.** The *save\_stack* is normally empty when a *TEX* program stops. But if, say, the user's input has an extra '{' (or a missing '}'), *TEX* will print the warning message

(\end occurred inside a group at level 1)  
(see §1335).

Explain in detail how to change *TEX* so that such warning messages will be more explicit. For example, if the source program has an unmatched '{' on line 6 and an unmatched '\begingroup' on line 25, your modified *TEX* should give two warnings:

```
(\end occurred when \begingroup
    on line 25 was incomplete)
(\end occurred when { on line 6
    was incomplete)
```

You may assume that *simple\_group* and *semi\_simple\_group* are the only group codes present on *save\_stack* when §1335 is encountered; if other group codes are present, your program should call *confusion*.

**34.** (The following question is the most difficult yet most important of the entire collection. It was the main problem on the take-home final exam.)

The purpose of this problem is to extend *TEX* so that it will sell better in China and Japan. The extended program, called *TEXX*, allows each font to contain up to 65536 characters. Each extended character is represented by two values, its 'extension' *x* and its 'code' *c*, where both *x* and *c* lie between 0 and 255 inclusive. Characters with the same '*c*' but different '*x*' correspond to different graphics; but they have the same width, height, depth, and italic correction.

*TEXX* is identical to *TEX* except that it has one new primitive command: \xchar. If \xchar occurs in vertical mode, it begins a new paragraph; i.e., it's a *(horizontal command)* as on p. 283 of *The T<sub>E</sub>Xbook*. If \xchar occurs in horizontal mode it should be followed by a *(number)* between 0 and 65535; this number can be converted to the form  $256x + c$ , where  $0 \leq x, c < 256$ . The corresponding extended character from the current font will be appended to the current horizontal list, and the space factor will be set to 1000. (If *x* = 0, the effect of \xchar is something like the effect of \char, except that \xchar disables ligatures and kerns and it doesn't do anything special to the space factor. Moreover, no penalty is inserted after an \xchar that happens to be the \hyphenchar of the current font.) A word containing an extended character will not be hyphenated. The \xchar command should not occur in math mode.

Inside *TEXX*, an extended character (*x, c*) in font *f* is represented by two consecutive *char\_node* items *p* and *q*, where we have *font(p) = null\_font*, *character(p) = qi(x)*, *link(p) = q*, *font(q) = f*, and *character(q) = qi(c)*. This two-word representation is used even when *x* = 0.

*TEXX* typesets an extended character by specifying character number  $256x + c$  in the DVI file. (See the *set2* command in §585.)

If *TEXX* is run with the macros of plain *TEX*, and if the user types '\tracingall \xchar600 \showlists', the output of *TEXX* will include

```
{\xchar}
{horizontal mode: \xchar}
{\showlists}

### horizontal mode entered at line 0
\hbox(0.0+0.0)x20.0
\tenrm \xchar"258
spacefactor 1000
```

(since 600 is "258 in hexadecimal notation).

Your job is to explain in detail *all* changes to *TEX* that are necessary to convert it to *TEXX*.

[Note: A properly designed extension would also include the primitive operator \xchardef, analogous to \chardef and \mathdef, because a language should be 'orthogonally complete'. However, this additional extension has not been included as part of problem 34, because it presents no special difficulties. Anybody who can figure out how to implement \xchar can certainly also handle \xchardef.]

**35.** The first edition of *TEX: The Program* suggested that extended characters could be represented with the following convention: The first of two consecutive *char\_node* items was to contain the font code and a character code from which the dimensions could be computed as usual; the second *char\_node* was a *halfword* giving the actual character number to be typeset. Fonts were divided into two types, based on characteristics of their TFM headers; 'oriental' fonts always used this two-word representation, other fonts always used the one-word representation.

Explain why the method suggested in problem 34 is better than this. (There are at least two reasons.)

◊ Donald E. Knuth  
Department of Computer Science  
Stanford University  
Stanford, CA 94305

## Philology

### Character Set Encoding

Nelson H.F. Beebe

#### Introduction

The article by Janusz S. Bień [3] which follows this paper complements an earlier one by Yannis Haralambous [5] on the subject of support for larger character sets.

Because this is an area of international interest in the computing community, it seemed worthwhile to review some of the issues, in order to provide background for those readers who are not actively following the subject.

There are currently at least two ISO groups that are actively engaged in the standardization of character set encoding. They are identified here by the reference numbers of the standards on which they are working, ISO 8859 and ISO 10646.

The ISO 8859 group deals with ASCII and EBCDIC character set issues and with standardization of 8-bit character sets. The ISO 10646 group deals with multi-byte character set issues.

I have been following the ISO 8859 work for more than two years, and recently joined the ISO 10646 discussions. Based on that experience, it seems clear that the problem is much more difficult than most people realize.

Both groups have active electronic mailing lists; the end of this article has information on how to subscribe to them.

#### 256 Does Not Suffice

There is a need for more than 256 characters to support even just those languages written in the Latin alphabet. As long as people (and computers) insist on using 8-bit characters, this gives rise to the problem of multiple 'code pages'.

Text encoded according to one code page must be accompanied by separate information stating what code page is to be used. This is difficult in attribute-free file systems such as UNIX and PC DOS, since there is no guaranteed way to keep that information with a text file. Embedding of attribute headers in the file itself is unacceptable.

Few electronic mail systems support the specification of a code page in the message header, although the Internet mail headers are sufficiently extensible that such support could be easily added.

Electronic mail is subject to character set translations, and these are often inconsistent, particularly if the mail has passed through Bitnet nodes or IBM mainframes; multiple code pages increase the likelihood of such corruption.

#### Switching Between Character Sets

Should it become necessary to switch code pages in the middle of a document (e.g. for a business in Sweden to address a letter to a customer in Turkey), some mechanism must be provided to do so. The ISO 8859 encodings of 8-bit characters define escape sequences that permit changing code pages.

For multi-byte character sets, the situation is more complex. JISCII [7, 8], the Japanese Industrial Standard Code for Information Interchange, is a 14-bit character set defined on a  $94 \times 94$  grid addressed by two 7-bit characters, using characters in the range 33...126, but biased downward by 32 so that the rows and columns are numbered from 1 to 94. The Chinese GB-2312 and Korean KS C 5601 standards also use a  $94 \times 94$  grid.

The JISCII character set includes special symbols and punctuation, the printable ISO/ASCII character set, Cyrillic, Greek, the Japanese syllabic alphabets (hiragana and katakana), followed by Level 1 kanji (2965 Chinese characters commonly used in Japanese), and Level 2 kanji (3388 lesser-used Chinese characters). JISCII does *not* include the ISO/ASCII control characters or European alphabetic extensions, nor does the ISO/ASCII subset occupy consecutive positions.

There are at least three ways of encoding documents in JISCII:

- 16-bit characters as 8-bit byte pairs;
- 7-bit ISO/ASCII with shift-in and shift-out escape sequences to enter and leave 16-bit character sections;
- 7-bit ISO/ASCII where character pairs whose first member has the high-order (8<sup>th</sup>) bit set are taken to be JISCII.

The last two are more compact than the first, but suffer from what may be called the *substring problem*.

Because these two involve a mixture of 8-bit and 16-bit characters, extraction of a valid substring requires examination of surrounding context. In the second method, it may be necessary to scan back to the start to determine whether there is a preceding escape sequence. In the third method, if the first character in the substring does not have its high bit set, one need only examine a single preceding

character to find out whether the first character is a normal one, or the second half of a pair.

Since string searching and substring extraction are among the commonest operations performed on text by a computer, these are very serious drawbacks.

There is also the problem of determining string lengths: is the length the number of characters, or the number of memory cells used to hold the string? Which one is needed depends on the application.

Use of a 16-bit representation eliminates these problems for JISCI, and could as well for a character set that supported all those derived from the Latin alphabet.

However, when Chinese is included, about 50,000 more characters are needed [4]. There are also differences in characters used in the People's Republic of China (due to simplifications instituted after 1949) and those in the Republic of China (Province of Taiwan).

When the 2800 syllabic characters of Korean Hangul are thrown in [4], plus the 900 or so letter variants of classical Arabic [9, 10, 2], and the dozens of writing systems used in India, it seems that even a 16-bit set of up to 65536 characters may be insufficient to cover the world's major languages. Because speakers of Chinese, Indian languages, and Arabic account for more than half of the world's population, these languages cannot be ignored.

Overlapping with the work on ISO 10646 is an effort to develop a comprehensive 16-bit character set called *Unicode*; some Unicode traffic was originally broadcast to the ISO 10646 list, but that practice was discontinued while this article was in preparation. Subscription details are given in the last section below.

The ISO 10646 list review contains the following paragraph:

As of March, 1990, two coding schemes have emerged. The International Organization for Standardization (ISO) Subcommittee 2, Working Group 2 (SC2/WG2) has developed the ISO 10646 Multi-Octet Code. It is now a "draft proposed" standard (two levels removed from being an international standard). The ISO working group has been working on this project for the last 6 years and it has been subject to unusually wide review for a proposed standard. The other draft standard is the result of the work of a consortium of U.S. companies, mostly from the west coast. It is called Unicode. Both of these draft standards enable the world's communication (newspapers and magazines) and busi-

ness characters, ideographs, and symbols to be encoded for storage and communication between computers. However, each uses a different approach to making the inevitable tradeoffs.

In my view, Unicode seems short-sighted, and too small. An 18-bit set would probably suffice, so maybe 36-bit machines like our venerable DEC-20, and the UNIVAC 1100 series, will someday be reincarnated! What is more likely, though, is that falling memory prices will make 32-bit characters practical.

### **Inadequate Display Support**

There is a serious problem of character display. How is a person to read a document that requires characters unavailable on the terminal or printing device? This becomes particularly relevant as we enter an era of international electronic mail and document exchange.

While personal computers and workstations are increasingly offering support for multiple character sets, much remains to be done before the display problem can be eliminated.

### **Impact on Programming Languages**

If character sets are enlarged, computer programming languages must be modified.

*TEX* 3.0 added only one bit to the character set encoding, and is riddled with assumptions that 256 is the size of the character set. These assumptions are of course introduced in the interests of compactness, so that *TEX* can run on small machines. With effort, some of these could be eliminated, but probably not all of them; doing so would introduce incompatibilities, and thus lose the right to the name *TEX*.

With the exception of the ANSI C standard [1], adopted in December 1989, existing programming languages (or at least their compilers) assume 7-bit or 8-bit characters; the last machines using only 6-bit characters were retired in the early 1980s.

ANSI C provides support for 'wide' characters; wide strings take the form L"..." and wide character constants are written as L'...'. Hexadecimal escape sequences, \xhhh... are introduced; they may have any number of hexadecimal digits. The underlying representation of wide character strings may use one or more bytes per character, allowing room for future expansion. Shift-in and shift-out representations are permitted. However, ANSI C states that a byte with all bits zero shall be interpreted as a null character (and therefore, a C string terminator), *independent of the shift state*, and a byte

with all bits zero may not occur in the second or subsequent bytes of a multibyte character. Also, a comment, string literal, or character constant shall begin and end in the initial shift state, and shall consist of a sequence of valid multibyte characters.

### Impact on Collating Sequences

Any assignment of characters to a numerical code introduces collating sequence problems.

For example, the Danish and Norwegian alphabets are A...Z, Æ, Ø, Å, while Swedish reverses the order of the last three and uses umlauts: Å, Ö, Ä.

Note that these Scandinavian accented letters are considered *separate* letters; this differs from French and German, which alphabetize such letters without regard to accents. Danish, Norwegian, and Swedish also occasionally use acute accents on the letters 'e' and 'o', for disambiguation of homonyms, and for a few foreign words; these accents are ignored in alphabetization.

With the orthography reform of 1948, Denmark ceased to capitalize nouns, introduced the new letter Å in place of the old Aa, and moved it from the front of the alphabet to the end. Under the reform, Aa is collated as if it were spelled Å, so some people moved from the front of the telephone book to the back. When Aa occurred in proper names, the owners were permitted to retain the old form, so both continue to exist: Aarhus University is in Århus, Denmark, and both the University and city listings are found at the end of the telephone book.

In German, ß ('scharfes s' or 'es-zet') capitalizes to SS (or rarely, SZ), does not occur as an initial letter, and is alphabetized as 'ss'.

In Spanish, 'ch' is treated as a single letter falling between c and d, 'll' is treated as a letter between l and m, and ñ is treated as a letter between n and o.

Although several languages in Eastern Europe and the Soviet Union employ the Cyrillic alphabet, there are variations between countries in both order, and the exact letters used. The reforms introduced after the Russian revolution in 1917 removed some letters from the alphabet, but scholars of pre-1917 literature still require them. A good treatment was given by David Birnbaum in a posting of 30-Nov-1989 to the ISO 8859 list.

The New York Stock Exchange listings are always by corporate abbreviations, yet collation is according to company name; IBM is listed as if it were spelled 'International Business Machines'. Telephone books in some areas move the Macdonalds and the McKays in front of other names beginning with M.

In Japanese and Chinese, the order of ideographic characters is determined by the authors of each dictionary. Many dictionaries base the order on the 214 fundamental 'radicals' (character part building blocks); the dictionary is ordered by groups of characters having the same radical, and within each group, by increasing numbers of strokes, and by pronunciation. However, some characters have more than one radical, many have the same pronunciation (in Japanese, 5500 kanji have only 336 different sounds [6]), and pronunciations may vary with dialects (Chinese has dozens of dialects that are mutually incomprehensible, but share a common writing system). Dictionaries from the People's Republic of China can also be found with ordering according to the Pinyin representation in the Latin alphabet, that is, according to Mandarin pronunciation.

JISCII has yet another assignment of Chinese (kanji) characters into two levels according to frequency of use. In the JIS Level 1 kanji, order is according to dictionary and pronunciation order [11, p. 68]; subgroupings are mostly according to stroke count, with exceptions. In JIS Level 2 kanji, order is according to radical and stroke count. These difficulties have traditionally discouraged the use of indexes in Japanese books, and also seriously impact filing of information in computers and offices.

For a readable account, see the chapter *Practical Consequences of a Large Character Set* in J. Marshall Unger's book [11].

Thus, in many languages, and even in English, sorting according to a collating sequence is a difficult problem, and capitalization cannot easily be changed by a computer program. This has important ramifications for BIBTEX, LATEX, and MakeIndex. In some BIBTEX styles, article titles are lower-cased, and some LATEX styles convert titles to uppercase letters; in both, the result is a disaster if the language happens to be German.

### Internationalization of Software

The chapters on Native Language Support and Regular Expressions in [12] describe the changes that must be made to the C run-time libraries, and to many UNIX utilities, when extended character sets are used.

For example, international software cannot contain embedded character strings; these must be moved into separate external files that can be customized for each language. In addition, output format strings must be extended syntactically to permit reordering of output tokens (cf. English "The White House" and French "La Maison Blanche").

## Summary and Conclusions

Character coding is a very complex issue, and despite the vigorous discussions on the ISO 8859 list, I do not see a solution on the horizon. Because it uses a different character set (EBCDIC) than everyone else, IBM will be affected more by character set issues than other vendors; its conservatism, and historical slowness to respond to the demands of the market and its users, also suggests that solutions will not soon be forthcoming.

In my view, the advent of support for 8-bit characters in TeX 3.0 will for some time hinder, rather than help, document portability. There is a conflict between the desire for ease of use and readability of the input file on the part of the author or typist who enters it by, say, striking the Ø key on a Danish keyboard, and the co-author in Britain who cannot display the Ø on the screen, and may have no idea what character was intended.

Authors who stick to the 7-bit ISO/ASCII character set and with some labor, enter \Ø{} instead of using the Ø key, will promote document portability.

Alternatively, translation filters will be needed, but it may not be possible to base them entirely on simple text substitutions, at least in the 7-bit to 8-bit direction, since \Ø cannot be substituted if it is the initial part of another control sequence. Also, in TeX 3.0, hyphenation opportunities will be lost if accented characters encoded as single 8-bit values are replaced by control sequences.

BIBTeX, LATEX, and MakeIndex will require revisions in the future for

- support of 8-bit character sets,
- more flexible provision for specification of sorting order,
- suppression of capitalization changes.

Even TeX itself may need modifications, since the `xchr[]` character translation array is initialized early in the program to values which depend upon the local character set, and no provision is made for switching code pages dynamically.

## Joining the Mailing Lists

To subscribe to the ISO 8859 or ISO 10646 mailing lists, send an e-mail message to the server

`LISTSERV@JHUVM.BITNET`

with the *body* text (LISTSERV ignores the e-mail *Subject:* line)

`SUBSCRIBE ISO8859 <your-personal-name>`  
or

`SUBSCRIBE ISO10646 <your-personal-name>`

Letter case is ignored in LISTSERV commands.

Your e-mail return address is automatically extracted from your mail message. The personal name is used to annotate the mailing list, which can be retrieved with a message like `REVIEW ISO10646`, in case you would like to know your correspondents by other than cryptic e-mail addresses. The `REVIEW` command also provides a summary of the purpose of the discussions.

All list traffic is archived; a message with the text `INDEX ISO10646` will retrieve an index for that list, and a following message with the text `GET ISO10646 filetype` will fetch a particular file. For more details on LISTSERV, send a message with the text `INFO GENINTRO`.

To get on the Unicode list, send a message requesting inclusion to Glenn Wright:

`glennw@sun.com`

## References

- [1] American National Standards Institute, 1430 Broadway, New York, N. Y., 10018. *American National Standard Programming Language C, ANSI X3-159.1989*, December 14 1989.
- [2] Joseph D. Becker. Arabic Word Processing. *Communications of the Association for Computing Machinery*, 30(7):600–610, July 1987.
- [3] Janusz S. Bień. On Standards for Computer Modern Font Extensions. *TUGboat*, 11(2):175–183, June 1990.
- [4] S. Duncan, T. Mukaii, and S. Kuno. A Computer Graphics System for Non-Alphabetic Orthographies. *Computer Studies in the Humanities*, 2(3):113–132, October 1969.
- [5] Yannis Haralambous. TeX and Latin Alphabet Languages. *TUGboat*, 10(3):342–345, November 1989.
- [6] A. V. Hershey. Calligraphy for Computers. Technical Report TR-2101, U. S. Naval Weapons Laboratory, Dahlgren, Virginia 22448, August 1967.
- [7] Japanese Standards Association, 1-24, Akasaka 4 Chome, Minato-ku, Tokyo, 107 Japan. *Japanese Industrial Standard JIS C 6626-1978 Code of the Japanese Graphics Character Set for Information Interchange*, 1978.
- [8] Japanese Standards Association, 1-24, Akasaka 4 Chome, Minato-ku, Tokyo, 107 Japan. *Japanese Industrial Standard JIS C 6234-1983 24-dots Matrix Character Patterns for Dot Printers*, 1983.

- [9] G.A. Kubba. The Impact of Computers on Arabic Writing, Character Processing, and Teaching. *Information Processing*, 80:961–965, 1980.
- [10] Pierre Mackay. Typesetting Problem Scripts. *Byte*, 11(2):201–218, February 1986.
- [11] J. Marshall Unger. *The Fifth Generation Fallacy—Why Japan is Betting its Future on Artificial Intelligence*. Oxford University Press, 1987.
- [12] X/Open Company, Ltd. *X/Open Portability Guide, Supplementary Definitions*, volume 3. Prentice-Hall, 1989.

◊ Nelson H.F. Beebe  
 Center for Scientific Computing  
 and Department of  
 Mathematics  
 South Physics Building  
 University of Utah  
 Salt Lake City, UT 84112  
 USA  
 Tel: (801) 581-5254  
 Internet: [Beebe@science.utah.edu](mailto:Beebe@science.utah.edu)

## On Standards for Computer Modern Font Extensions

Janusz S. Bień

### Abstract

Haralambous' proposal to standardize the unused part of Computer Modern fonts is discussed, and some modifications and extensions suggested. The idea is pursued by designing the extended CM font layout, and an example is given for one of its possible uses.

### 1 Introduction

In my note [4] I advocated an old ([15, p. 46], [6, p. 45]) but rarely used idea to place national letters (actually, the Polish ones, but the generalization is obvious) in the unused part of Computer Modern fonts, i.e. as the characters with the codes higher than 127; this approach allows the handling of national languages in a way upward compatible with the standard (American) English *TEX*. A similar proposal was made independently by Yannis Haralambous [8], who states also that the use of non-English letters of latin alphabets should be coordinated, resulting in a single widely used extension

to Computer Modern fonts — I strongly support the principal idea, and I pursue it in the present paper. To organize the discussion in a systematic way, I will use the notions — borrowed from [2] — of *text encoding*, *typing* and *rendering*.

### 2 Text encoding

In the context of *TEX*, *encoding* means the character sets of the fonts in question and their layouts. In the present section I will focus my attention on the character sets, as the layouts should be influenced, among others, by *typing* considerations.

In an attempt to obtain a general idea about the use of the latin alphabet worldwide, I looked up the only relevant reference work I am aware of, namely *Languages Identification Guide* [7] (hereafter *LIG*). Apart from the latin scripts used in the Soviet Union and later replaced by Cyrillic ones, it lists 82 languages using the latin alphabet with additional letters (I preserve the original spelling):

Albanian, Aymara, Basque, Breton, Bui, Catalan, Choctaw, Chuana, Cree, Czech, Danish, Delaware, Dutch, Eskimo, Esperanto, Estonian, Ewe, Faroese (also spelled Faroeish), Fiji, Finnish, French, Frisian, Fulbe, German, Guarani, Hausa, Hungarian, Icelandic, Irish, Italian, Javanese, Juang, Kasubian, Kurdish, Lahu, Lahuli, Latin, Lettish, Lingala, Lithuanian, Lisu, Luba, Madura, Miao, Malagash, Malay, Mandingo, Minankabaw, Mohawk, Mossi, Navaho, Norwegian, Occidental, Ojibway (also spelled Ojibwe), Polish, Portuguese, Quechua, Rhaeto-Romanic (Ladin, Romansh), Rumanian, Samoan, Seneca, Serbo-Croatian, Sioux, Slovak, Slovene, Spanish, Suto, Sundanese, Swahili, Swedish, Tagalog, Turkish, Uolio, Vietnamese, Volapük, Welsh, Wolof, Y, Yoruba, Zulu.

This list includes some languages and dialects with no script at all, for which the information supplied concerns more or less standard transcription. For most of them this fact is noted explicitly, but the exception of Kasubian (usually recognized as a dialect of Polish) suggests that this is not always the case. I noticed some inconsistencies in the numerous indexes to the book, but only one omission (described later) in the proper text. Of course, it is difficult for me to judge the reliability of the work as a whole.

The number of additional letters in the latin alphabets listed in *LIG* — including some variants of shape but excluding upper case letters — is 176.

Hence the total number of lower and upper case letters is definitely over 300. The possible errors and omissions cannot change this estimate significantly, so in general we have to cope with the number of additional letters substantially exceeding the number of free slots in the Computer Modern fonts.

My solution to this problem is to postulate two levels of standards:

**Extended Computer Modern fonts**, with a small number of slots unassigned.

**Full Extended Computer Modern fonts**, i.e. national or regional fonts compatible with Extended CM fonts, but having some additional characters assigned.

Of course, both of them will include all the characters of the original CM fonts in their proper places; although teletypewriter layout fonts are much less used, our standards should take them into account, too.

It should be noted now that there are numerous national and international standards for text encoding. The most relevant for us is the ISO 6937 international standard ([12], [13], [14]), described thoroughly in [25] and discussed in [24]. Annex D to the standard [13] is entitled *Use of Latin alphabetic characters*; formally it is not part of the standard, but its goal is to provide

justification for the composition of the alphabetic part of the graphic character repertoire. It does not attempt to define which characters should, and which ones should not, be used in any language.

The annex contains a table (quoted in [25]) listing the following languages (I again preserve the original spelling):

Albanian, Basque, Breton, Catalan, Croat, Czech, Danish, Dutch, English, Estonian, Faroese, Finnish, French, Frisian, Galician, German, Greenlandic, Hungarian, Icelandic, Irish, Italian, Lapp, Latvian, Lithuanian, Maltese, Norwegian, Occitan, Polish, Portuguese, Rhaeto-Romanic, Romanian, Scots Gaelic, Slovak, Slovene, Sorbian, Spanish, Swedish, Turkish, Welsh, Afrikaans, Esperanto.

With the exception of the last 2 languages, the list contains 39 living European languages. However, despite the quoted reservation, it seems rather strange that, according to the table, English uses 28 additional letters (namely á Á, à À, æ Æ, ç Ç, é É, è È, ê Ê, ë Ë, î Î, ï Ï, ñ Ñ, ô Ô, ö Ö, œ œ). The standard associates with all the characters their

unique identifications (explained in Annex A to the standard [13]) and names; I will use these names in the sequel when appropriate.

The ISO 6937 character set includes 87 additional letters which exist in both lower and upper case form, 6 letters which have only lower case form and 2 letters which have only upper case form. Additionally, 3 lower case letters and 1 upper case letter have shape variants (I refer here to the shapes of the letters, not to their function in specific languages; although e.g. in Lapp and Latvian G is the upper case equivalent of g, I count them as having no case counterparts). This gives us the total of 186 additional letters. Although 10 of them are already included in the original CM fonts (namely æ Æ, ı ı, l L, œ œ, ø Ø, ß ß), again the number of additional characters exceeds the number of free slots. Moreover, we should not forget the problem of the missing punctuation marks. The most demanded ones seem to be the angle quotation marks («,») used also e.g. in French, German and Polish, the "continental" left quotation mark („) used e.g. in German and Polish, and perhaps the German right quotation mark („); cf. [6], [21], [18].

Let us have now a closer look at the character set proposed by Haralambous. To understand fully its implications, let us discuss first the language list contained in [8]. The ISO standard and *LIG* confirm consistently only 8 items:

**Croat** (spelled Croatian in [8] and [7]): č Č, đ Đ, š Š, ž Ž.

**Hungarian**: á Á, é É, í Í, ó Ó, ö Ö,ő Ő, ú Ú, ü Ü, ū Ÿ.

**Polish** (in addition, I vouch for its correctness personally): ą A, ć Ć, ę Ę, ł Ł, ń Ñ, ó Ó, ś Š, ž Ž, ż Ż.

**Romanian** (spelled Rumanian in [7]): â Â, ă Â, î Î, ş ř, t T.

**Slovene** (spelled Slovenian in [7] and [8]): č Č, š Š, ž Ž.

**Spanish**: á Á, é É, í Í, ñ Ñ, ó Ó, ú Ú, ü Ü.

**Turkish**: â Â, ç Ç, ğ Ğ, ı İ, ì î, ì î, ö Ö, ş ř, û Ü, ü Ü.

In the case of 7 languages my sources consistently disagree with Haralambous' list:

**Albanian**. There is c with cedilla (ç Ç) instead of c with caron (č Č).

**Catalan**. There is the additional letter i with diaeresis (í); according to ISO 6937, there is an additional letter l with middle dot, while *LIG* states

Two successive letters l which do not denote one sound are separated by a point l.l (or l·l).

**Czech.** The letter d' is treated as variant of ð; both of them are called in ISO 6937 *small d with caron*; the same holds respectively for t'. *LIG* distinguishes also a variant of ð differing in the placement of the caron. For upper case letters both sources list only Ď and Č (neither D' nor T').

**Faroese** (in [7] often spelled Faroeish). Instead of *small d with stroke* and *small thorn* there should be *small eth* (ð)<sup>1</sup> and *capital D with stroke* (i.e. the capital eth Đ).

**Icelandic.** Instead of *small d with stroke* there should be *capital thorn* (Þ).

**Irish.** Besides its own alphabet, the language uses the latin script with the following additional letters: á Á, é É, í Í, ó Ó, ú Ú.

**Lithuanian.** It uses *ogonek* instead of *cedilla*, so there is e.g. à and è instead of ã and ç, etc.

For the remaining 28 languages, 9 languages are not accounted for in the ISO 6937 standard (Corsican, Creole, Gaelic, Guarani, Indonesian, Kurdish, Latin, Qhëshwa, Vietnamese) and 7 languages are not listed in *LIG* (Corsican, Creole, Gaelic, Galician, Maltese, Occitan, Qhëshwa); however, some languages may be called by different names (I happen to know that Latvian is Lettish, but is Scots Gaelic different from Gaelic, is Qhëshwa different from Quechua?). For the rest of them both my sources more or less disagree. Fortunately, with the exception of Slovak and Vietnamese, the differences concern the use of accented letters by specific languages and do not affect the character set itself.

For Slovak (spelled Slovakian by Haralambous), the problem concerns the letter l with acute accent included in the ISO standard but not listed at all in *LIG*, and the letter l with caron, listed in *LIG* (and by Haralambous) only in its variant shape (l'). I consulted an original Slovak grammar [22], which confirms the existence of ľ and Ľ and lists l with caron only in the form l' L'.

As for Vietnamese, *LIG* (and also some books published in Poland) uses o' and u' instead of ö and ü (o and u "with beard"), listed not only by Haralambous but also in [26]; on the other hand, there is no doubt about the correct shape of the accent called question mark in [26], which is given

<sup>1</sup> The editors thank Jörgen Pind for supplying his METAFONT sources (see also [23]) to create the eths and thorns in this article.

by Haralambous in a simplified form. I intend to consult an expert on this matter (I suspect different usage in North and South Vietnam), but his answer is not relevant for our further discussion—anyway, the Vietnamese letters and accents should be included in a specific Full Extended CM font, not in the Extended CM font.

In my opinion, the Extended CM fonts should contain the following additional letters:

- small and capital a with acute (á Á), grave (à Ä) and circumflex (â Å) accent, with diaeresis (ä Å), tilde (ã Ä), ring (å Å) and ogonek (ą Å),
- small and capital c with acute (ć Ć) accent, with cedilla (ç Ç) and with caron (č Č),
- small and capital d with caron (đ Đ) and with stroke (đ, Đ),
- small eth (ð), small and capital thorn (þ Þ),
- small and capital e with acute (é É), grave (è È) and circumflex (ê Ê) accent, with diaeresis (ë Ë) and ogonek (ë Ë),
- small and capital g with breve (ğ Ĝ),
- small and capital i with acute (í Í), grave (í ï) and circumflex (í î) accent, with diaeresis (í î) and caron (í ī), and capital I with dot above (í ī),
- small and capital l with acute accent (í Ł), with caron (ł Ł) and with stroke (ł Ł),
- small and capital n with acute accent (ń Ñ), with tilde (ñ ñ) and caron (ń ñ),
- small and capital o with acute (ó Ó), grave (ò Ö) and circumflex (ô Ô) accent, with diaeresis (ö Ö) and caron (ö Ö), and with double acute accent (ő Ö),
- small and capital r with caron (ř Ř),
- small and capital s with acute (š Š) accent, with cedilla (ş ş) and with caron (š Š),
- small and capital t with cedilla (ť Ť) and with caron (ť Ť),
- small and capital u with acute (ú Ú), grave (ù Ü) and circumflex (û Ü) accent, with diaeresis (ü Ü), ring (û Ü) and with double acute accent (ű Ü),
- small and capital y with acute (ý Ý) accent and with diaeresis (ÿ ÿ),
- small and capital z with acute (ž Ž) accent, with caron (ž Ž) and with dot above (ž Ž).

and the following additional punctuation marks:

- the left and right angle quotation marks (« »),
- the "continental" left quotation mark („ „),
- the German right quotation mark (" ") .

The proposed character set thus contains 112 additional letters and 4 additional punctuation marks. It includes the Polish letters Ł Ł, already present in some CM fonts, because they are needed also in the fonts with the teletypewriter layout (I follow Haralambous in this respect).

The Extended Computer Modern font leaves 12 slots to be assigned in the regional or national Full Extended CM fonts (in particular, for Vietnamese).

### 3 Text typing

In my note [4] I advocated a novel idea (at least at that time — now cf. [27, p. 335]) to use several *tfm* files to access the same font for different purposes — a Polish font with the layout upward compatible with the original CM font can be accessed by the original *tfm* for standard work, and by a special *tfm* file for typesetting Polish texts. In my opinion, this approach should be applied to the multilingual fonts discussed here — they should be offered with many *tfm* files tailored for specific regions, nations and languages. Therefore in the sequel I will limit my attention to the *default tfm* files for Extended CM fonts.

In general, the typing considerations have two aspects

- echo problem,
- sorting problem.

By the echo problem I mean the typing feedback — can the user pressing a key on the keyboard see the proper character shape on the screen without resorting to the graphic mode? As for the sorting problem, many people are not aware that the alphabetic ordering is language dependent, and that it can differ substantially from one language to another. Of course, *TEX* users are first of all interested in sorting by various *TEX* utilities, such as *BIBTEX* or *MakeIndex*. I hope that the re-implementation of *LATEX* proposed in [20] will be accompanied by the universal versions of these programs, allowing the sorting algorithm to be controlled by appropriate parameters.

Unfortunately, the echo problem is not an internal affair of the *TEX* community, but a general problem heavily dependent on hardware and operating systems. As mentioned in [9], over half of *TEX* users work on IBM compatible computers, so it would not be wise to ignore what IBM intends to do in this domain. Therefore I have done my best to collect the tables of the so called *code pages* designed by IBM (or with its approval).

In [19] I found the following tables:

1. Code page 437 — United States,

2. Code page 850 — Multilingual,
3. Code page 860 — Portuguese,
4. Code page 863 — French-Canadian,
5. Code page 865 — Nordic.

Surprisingly enough, there were mistakes in the tables; I managed to correct them by consulting other sources.

In [10] I found, apart from Cyrillic, the following page

1. Code page 852 — Multilingual Group 2.

In [11] I found, apart from Cyrillic and 22 EBCDIC-based pages, the following code pages (for the curious reader I include also non-latin scripts):

1. Code page 838 — Latin #5, Thailand,
2. Code page 850 — Multinational,
3. Code page 851 — Greece,
4. Code page 857 — Latin #5, Turkey,
5. Code page 860 — Portugal,
6. Code page 861 — Iceland,
7. Code page 862 — Israel,
8. Code page 863 — Canadian French,
9. Code page 864 — Arabic,
10. Code page 865 — Nordic,
11. Code page 891 — Korea,
12. Code page 897 — Japan #1,
13. Code page 903 — Peoples Republic of China (PRC),
14. Code page 904 — Republic of China (ROC).

As you can see, the page names differ slightly in various documents.

My goal was to design the layout of Extended CM fonts in a way as compatible as possible with the above listed code pages. I think that seeing on the screen — instead of a letter — a non-letter character is less confusing than seeing a wrong letter; therefore I looked first of all for those letters which appear in at least two code pages and which conflict only with some non-letter characters. I found 8 such letters, and I included them in the font on the positions identified by their codes in the code pages (the octal values are given in parentheses):

*small A with circumflex accent (â) 131 ('203),*  
*small c with cedilla (ç) 135 ('207),*  
*capital C with cedilla (Ç) 128 ('200),*  
*small e with acute accent (é) 130 ('202),*  
*small o with acute accent (ó) 162 ('242),*  
*small o with circumflex accent (ô) 147 ('223),*  
*small u with diaeresis (ü) 129 ('201),*  
*capital U with diaeresis (Ü) 154 ('232).*

I decided also to prefer those code pages which are provided now with MS-DOS and PC-DOS,

namely the pages 437 and 850. So the second step was to include those letters which occur in both of them, and those which occur in page 850 and are in conflict only with non-letter characters in page 437. It resulted in the following 49 assignments.

*small a with acute accent (á) 160 ('240),  
 capital A with acute accent (Á) 181 ('265),  
 small a with grave accent (à) 133 ('205),  
 capital A with grave accent (À) 183 ('267),  
 capital A with circumflex accent (Â) 182  
 ('266),  
 small a with diaeresis (ä) 132 ('204),  
 capital A with diaeresis (Ä) 142 ('216),  
 small a with tilde (ã) 198 ('306),  
 capital A with tilde (Ã) 199 ('307),  
 small a with ring (å) 134 ('206),  
 capital A with ring (Å) 143 ('217),  
 small eth (ð) 208 ('320),  
 capital D with stroke (Ð) 209 ('321),  
 small thorn (þ) 232 ('350),  
 capital thorn (Þ) 231 ('347),  
 capital E with acute accent (É) 144 ('220),  
 small e with grave accent (è) 138 ('212),  
 capital E with grave accent (È) 212 ('324),  
 small e with circumflex accent (ê) 136 ('210),  
 capital E with circumflex accent (Ê) 210 ('322),  
 small e with diaeresis (ë) 137 ('211),  
 capital E with diaeresis (Ë) 211 ('323),  
 small i with acute accent (í) 161 ('241),  
 capital I with acute accent (Í) 214 ('326),  
 small i with grave accent (í) 141 ('215),  
 capital I with grave accent (Ì) 222 ('336),  
 small i with circumflex accent (î) 140 ('214),  
 capital I with circumflex accent (Î) 215 ('327),  
 small i with diaeresis (ï) 139 ('213),  
 capital I with diaeresis (Ï) 216 ('330),  
 small n with tilde (ñ) 164 ('244),  
 capital N with tilde (Ñ) 165 ('245),  
 capital O with acute accent (Ó) 224 ('340),  
 small o with grave accent (ò) 149 ('225),  
 capital O with grave accent (Ò) 227 ('343),  
 capital O with circumflex accent (Ô) 226  
 ('342),  
 small o with diaeresis (ö) 148 ('224),  
 capital O with diaeresis (Ö) 153 ('231),  
 small o with caron (ô) 228 ('344),  
 capital O with caron (Õ) 229 ('345),  
 small u with acute accent (ú) 163 ('243),  
 capital U with acute accent (Ú) 233 ('351),  
 small u with grave accent (ù) 151 ('227),  
 capital U with grave accent (Ù) 235 ('353),  
 small u with circumflex accent (û) 150 ('226),  
 capital U with circumflex accent (Û) 234  
 ('352),*

*small y with acute accent (ý) 236 ('354),  
 capital Y with acute accent (Ý) 237 ('355),  
 small y with diaeresis (ÿ) 152 ('230).*

This rule applies also to the punctuation marks:

*left angle quotation mark («) 174 ('256),  
 right angle quotation mark (») 175 ('257).*

The next step was to transfer to our font the letters included only in the second multinational page, namely 852, and not in conflict with some letter in other pages, i.e. the following letters:

*capital N with caron (Ñ) 213 ('325),  
 small r with caron (ř) 253 ('375),  
 capital R with caron (Ř) 252 ('374),  
 capital S with cedilla (Š) 184 ('270),  
 capital S with caron (Š) 230 ('346),  
 small t with cedilla (ť) 238 ('356),  
 capital T with cedilla (Ť) 221 ('335),  
 small t with caron (ť) 156 ('234),  
 small u with double acute accent (ü) 251  
 ('373),  
 small z with acute accent (ź) 171 ('253),  
 small z with dot above (ż) 190 ('276),  
 capital Z with dot above (Ž) 189 ('275).*

By this time we have filled in 71 slots in the font; 12 slots are to be left free and 45 characters are still to be assigned. It is the right moment to concentrate on the free slots. I decided to leave free the positions 145 ('221), 146 ('222), 155 ('233) and 157 ('235), because in the most used page, 437, they contained the characters æ Æ ø Ø, which can be useful for many TeX users. For similar reasons I left free the position 225 ('341), which in the two popular pages 850 and 852 (and also 857) contain the character þ. I decided also to leave free the positions 159 ('237), 166 ('246), 167 ('247), 168 ('250), 169 ('251), 172 ('254) and 173 ('255), because I see no simple criterion for solving the letter conflicts among the code pages. There are also serious conflicts on the positions 158 and 170, so I decided to devote them to the punctuation marks:

*the "continental" left quotation mark („) 158  
 ('236),*

*the German right quotation mark (“) 170  
 ('252).*

The remaining 43 characters have been assigned in an arbitrary way:

*small a with ogonek (ą) 176 ('260),  
 capital A with ogonek (Ą) 177 ('261),  
 small a with breve (ă) 178 ('262),  
 capital A with breve (Ă) 179 ('263),  
 small c with acute accent (ć) 180 ('264),  
 capital C with acute accent (Ć) 185 ('271),*

small c with caron (č) 186 ('272),  
 capital C with caron (Č) 187 ('273),  
 small d with caron (đ) 188 ('274),  
 capital D with caron (Đ) 191 ('277),  
 small d with stroke (đ) 192 ('300),  
 small e with ogonek (ę) 193 ('301),  
 capital E with ogonek (Ę) 194 ('302),  
 small e with caron (ě) 195 ('303),  
 capital E with caron (Ě) 196 ('304),  
 small g with caron (ǵ) 197 ('305),  
 capital G with caron (Ѓ) 200 ('310),  
 small i with caron (í) 201 ('311),  
 capital I with caron (І) 202 ('312),  
 capital I with dot above (і) 203 ('313),  
 small l with caron (љ) 204 ('314),  
 capital L with caron (Љ) 205 ('315),  
 small l with acute accent (љ) 206 ('316),  
 capital L with acute accent (Љ) 207 ('317),  
 small l with stroke (љ) 217 ('331),  
 capital L with stroke (Љ) 218 ('332),  
 small n with acute accent (ń) 219 ('333),  
 capital N with acute accent (Ń) 220 ('334),  
 small n with caron (њ) 223 ('337),  
 small o with double acute accent (ő) 239 ('357),  
 capital O with double acute accent (Ő) 240  
 ('360),  
 small s with acute accent (ś) 241 ('361),  
 capital S with acute accent (Ś) 242 ('362),  
 small s with cedilla (ş) 243 ('363),  
 small s with caron (š) 244 ('364),  
 capital T with caron (Ҭ) 245 ('365),  
 small u with ring (ú) 246 ('366),  
 capital U with ring (Ӯ) 247 ('367),  
 capital U with double acute accent (Ӱ) 248  
 ('370),  
 capital Y with diaeresis (Ӳ) 249 ('371),  
 capital Z with acute accent (Җ) 250 ('372),  
 small z with caron (ž) 254 ('376),  
 capital Z with caron (Җ) 255 ('377).

Editor's note: The encoding scheme above is presented in a font layout on p. 183.

The default **tfm** files for Extended CM fonts for use with 8-bit **TeX** should not contain any ligatures except those needed for kerning or inherited from the original CM fonts. However, for 7-bit **TeX** another *default tfm* scheme is to be designed, because in it, ligatures are the only way to access the second half of the fonts without disturbing the hyphenation. I would like to advocate here another idea from my note [4], consisting in using the character with the code 32 (the stroke for the Polish l) as a part of the ligatures accessing the national letters. The idea is further developed here in two respects:

- The ligatures in question should consist of a letter *followed* by the character 32. The reason is that such representation of national letters affects the alphabetic ordering in a less substantial way and, under some additional conditions, can even preserve the ordering for some languages.
- There should be a general rule saying that the ligature composed of a character with the code *x* followed by the character with the code 32 accesses the character with the code *x* + 128. The rule can be called a 7-bit equivalent of the double circumflex notation [16, p. 325].

Of course, the character 32 is not directly accessible, because it coincides with the space character in the ASCII code. However, it can be easily assigned to any active character. On the other hand, to preserve the compatibility in case of the teletype layout fonts, the macro for the visible space has to be changed.

I think that the language specific **tfm** files are especially useful for 7-bit **TeX**. My experience with typesetting Russian texts using the AMS Cyrillic fonts showed that sophisticated multipurpose ligature tables are more a nuisance than a real help. In consequence, Haralambous' ligatures can be accepted only as one of several alternative **tfm** files, and not as a general standard.

#### 4 Text rendering

In the context of **TeX**, *rendering* means the actual fonts used by the device drivers. Again, in my opinion, there should be a *default METAFONT* definition, not the standard one. First, I am not sure that e.g. French *capital A with acute accent* looks the same as the Hungarian one (my impression — maybe wrong — is that they differ substantially). Secondly, I do not know whether such problems as the actual shape of e.g. Czech *d with caron* can be solved definitively; perhaps both versions are to be used depending on the situation.

Last but not least, it should be remembered that some letters use up the font space only for hyphenation purposes — even in **TeX** 3.0 an accented letter (i.e. constructed by the \accent command) disables hyphenation until the next glue. Sooner or later a standard for *virtual fonts* — i.e. for creating new characters from the elements already present in the fonts — will emerge as a part of the standardization of the device drivers. One of the first virtual font mechanisms was mentioned in the Editor's comment to Haralambous' paper ([8, p. 342]), but the idea of "fooling" the **TeX** program can be traced down at least to Appelt [1]. Incidentally, the

term *virtual fonts* is used in the context of Beebe's drivers in a totally different sense—cf. the 'a' parameter ([3, p. 3]); I hope this confusing use will soon be abandoned.

## 5 An example

Let us imagine an IBM PC computer equipped with the code page 852 character set (supported by IBM on the Polish market and accepted by some state-owned manufacturers), used to typeset Polish texts with 8-bit *TeX* and the proposed Extended CM fonts. There are 18 Polish national letters, and only for 4 of them their codes coincide in the code page and the proposed layout. In consequence, some kind of translation is needed for the remaining 14 letters (such a compromise seems necessary to make the proposal acceptable by the users of other languages).

Assuming that the fonts have been set up correctly (by assigning to their characters the proper values of `\catcode`, `\lccode`, `\uccode`, `\sfcode`, `\mathcode` and `\delcode`), the following definitions are sufficient for the compatibility of the echo (when working with a standard 8-bit editor) with the font layout.

```
% 165 small a with ogonek
\catcode^^a5=\active\chardef^^a5=176
% 164 capital A with ogonek
\catcode^^a4=\active\chardef^^a4=177
% 134 small c with acute accent
\catcode^^86=\active\chardef^^86=180
% 143 capital C with acute accent
\catcode^^8f=\active\chardef^^8f=185
% 169 small e with ogonek
\catcode^^a9=\active\chardef^^a9=193
% 168 capital E with ogonek
\catcode^^a8=\active\chardef^^a8=194
% 136 small l with stroke
\catcode^^88=\active\chardef^^88=217
% 157 capital L with stroke
\catcode^^9d=\active\chardef^^9d=218
% 228 small n with acute accent
\catcode^^e4=\active\chardef^^e4=219
% 227 capital N with acute accent
\catcode^^e3=\active\chardef^^e3=220
% 224 capital O with acute accent
\catcode^^e0=\active\chardef^^e0=224
% 152 small s with acute accent
\catcode^^98=\active\chardef^^98=241
% 151 capital S with acute accent
\catcode^^97=\active\chardef^^97=242
% 141 capital Z with acute accent
\catcode^^8d=\active\chardef^^8d=250
% no translation needed for
% 162 small o with acute accent
```

```
% 171 small z with acute accent
% 190 small z with dot above
% 189 capital z with dot above
```

After changing the representation of Polish letters in the hyphenation patterns [17], the Polish hyphenation algorithm will operate with no problems.

As for 7-bit *TeX*, using directly the default 7-bit *tfm* would make the input text completely unintelligible. However, it is not difficult to create a convenient interface, either by means of macro definitions similar to those quoted in [18, p. 5] and [5], or by introducing a special Polish *tfm* file with appropriate ligatures.

In both cases the explicit use of national letters (i.e. echoed on the screen in a reasonable way) in control sequences is severely limited. Unfortunately, we have to live with it till the next change in *TeX*.

## 6 Concluding remarks

For a standard to be widely accepted, it has to be fully adequate to actual needs—neither too general nor too specific. I hope that my modifications and extensions of Haralambous' proposal achieve the proper balance.

It should be also noted that a substantial part of actual and potential *TeX* users who will be affected by the standards are not yet organized into users groups; moreover, most of them have no access to electronic mail. If the standard is to be developed—as proposed by Haralambous—in a democratic way, then the traditional forms of communication should be the primary medium.

## References

- [1] Wolfgang Appelt. The Hyphenation of Non-English Words with *TeX*. In Dario Lucarella, editor, *Proceedings of the First European Conference on *TeX* for Scientific Documentation*, Addison-Wesley, Reading, Massachusetts, 1985, pp. 61–65.
- [2] Joseph D. Becker. Multilingual Word Processing. *Scientific American* Vol. 251 No. 1 (July 1984), pp. 82–93.
- [3] [Nelson H. F. Beebe]. *DVIxxx—Display *TeX* DVI Files on Assorted Output Devices*. Beebe's driver distribution version 2.10.
- [4] Janusz S. Bień. Polish Language and *TeX*. *TEXline* 8 (January 1989), p. 2.
- [5] Janusz S. Bień. Co to jest *TeX*? [What is *TeX*? In Polish]. *Wiadomości Matematyczne* Vol. 29 No. 1 (to appear).
- [6] Jacques Désarménien. The Use of *TeX* in French: Hyphenation and Typography. In

- Dario Lucarella, editor, *Proceedings of the First European Conference on T<sub>E</sub>X for Scientific Documentation*, Addison-Wesley, Reading, Massachusetts, 1985, pp. 41–59.
- [7] R. S. Gilyare and V. S. Grivnin. *Languages Identification Guide*. “NAUKA” Publishing House, Central Department of Oriental Literature, Moscow 1970.
  - [8] Yannis Haralambous. T<sub>E</sub>X and latin alphabet languages. *TUGboat* Vol. 10 No. 3 (November 1989), pp. 342–345.
  - [9] Don Hosek. Guidelines for creating portable METAFONT code. *TUGboat* Vol. 10 No. 2 (July 1989), pp. 173–176.
  - [10] IBM Corporation. *Personal System/2 Natural Language Supplement*. First edition (February 1988) 07F3226.
  - [11] IBM Corporation. *Application System/400 Natural Language Support: User's Guide*. First edition (September 1989) GC21-9877-0.
  - [12] International Organization for Standardization. *Information processing. Coded character sets for text communication—Part 1: General Introduction*. First edition 1983-11-01. Ref. No. ISO 6937/1-1983(E).
  - [13] International Organization for Standardization. *Information processing. Coded character sets for text communication—Part 2: Latin alphabetic and non-alphabetic graphic characters*. First edition 1983-12-15. Ref. No. ISO 6937/2-1983(E).
  - [14] International Organization for Standardization. *Information processing. Coded character sets for text communication—Part 2: Latin alphabetic and non-alphabetic graphic characters*. Addendum 1, 1989-05-01. Ref. No. ISO 6937-2-1983/Add1:1989(E).
  - [15] Donald E. Knuth. *The T<sub>E</sub>Xbook*. Addison-Wesley, Reading, Massachusetts, 1984.
  - [16] Donald E. Knuth. The new versions of T<sub>E</sub>X and METAFONT. *TUGboat* Vol. 10 No. 3 (November 1989), pp. 325–328.
  - [17] Hanna Kołodziejska. Dzielenie wyrazów polskich w systemie T<sub>E</sub>X [Polish hyphenation patterns for T<sub>E</sub>X; in Polish]. IInf UW Report 165, Institute of Informatics, Warsaw University, 1987.
  - [18] Hanna Kołodziejska. Le traitement des textes polonais avec le logiciel T<sub>E</sub>X. *Cahiers GUTenberg Numéro zéro* (Avril 1988), pp. 3–10.
  - [19] Microsoft Corporation. *MS-DOS User's Guide and User's Reference* [Version 3.3]. Doc. No. M5123-8806B.
  - [20] Frank Mittelbach and Rainer Schöpf. With L<sub>A</sub>T<sub>E</sub>X into the Nineties. *TUGboat* Vol. 10 No. 4 (December 1989), pp. 681–690.
  - [21] Hubert Partl. German T<sub>E</sub>X. *TUGboat* Vol. 9 No. 1 (April 1988), pp. 70–72.
  - [22] Eugen Pauliny. Krátká gramatika slovenská. Slovenské Pedagogické Nakladatel'stvo, Bratislava 1963.
  - [23] Jörgen L. Pind. Lexicography with T<sub>E</sub>X. *TUGboat* Vol. 10 No. 4 (December 1989), pp. 655–665.
  - [24] Staffan Romberger and Yngve Sundblad. Adapting T<sub>E</sub>X to languages that use Latin alphabetic characters. In Dario Lucarella, editor, *Proceedings of the First European Conference on T<sub>E</sub>X for Scientific Documentation*, Addison-Wesley, Reading, Massachusetts, 1985, pp. 27–40.
  - [25] Joan M. Smith. Transmitting Text: A Standard Way of Communicating Characters (Part 1). *Association for Literary and Linguistic Computing Bulletin* Vol. 12 (1983) No. 2, pp. 11–38.
  - [26] Eric Vogel. Printing Vietnamese characters by adding diacritical marks via T<sub>E</sub>X. *TUGboat* Vol. 10 No. 2 (July 1989), pp. 217–221.
  - [27] Dimitri Vulis. Notes on Russian T<sub>E</sub>X. *TUGboat* Vol. 10 No. 3 (November 1989), pp. 332–336.
- ◊ Janusz S. Bień  
Institute of Informatics  
Warsaw University  
PKiN p.850  
00-901 Warszawa, Poland

	'0	'1	'2	'3	'4	'5	'6	'7	
'20x	Ç	ü	é	â	ä	à	å	ç	"8x
'21x	ê	ë	è	ï	î	ì	Ä	À	"9x
'22x	É	free	free	ô	ö	ò	û	ù	"Ax
'23x	ÿ	Ö	Ü	free	ť	free	„	free	"Bx
'24x	á	í	ó	ú	ñ	Ñ	free	free	"Cx
'25x	free	free	“	ž	free	free	«	»	"Dx
'26x	ą	Ą	ă	Ă	ć	Á	Â	À	"Ex
'27x	Ş	Ć	č	Č	đ	Ž	ż	Đ	"Fx
'30x	đ	ę	Ę	ě	Ě	gó	ã	Ã	
'31x	Ğ	ĩ	Ĩ	İ	İ	Ł	í	Ĺ	
'32x	ð	Đ	Ê	Ë	È	Ñ	Í	Î	
'33x	İ	ł	L	ń	Ń	Ŧ	Ì	ň	
'34x	Ó	free	Ô	Ò	ő	Ó	Š	P	
'35x	þ	Ú	Û	Ù	ý	Ý	ť	ő	
'36x	Ö	ś	Ś	ş	š	Ŧ	ü	Ü	
'37x	Ü	Ŷ	Ž	ú	Ř	ř	ž	Ž	
	"8	"9	"A	"B	"C	"D	"E	"F	

## The layout of the proposed CM font extensions

Janusz S. Bień



### Circular Reasoning: Typesetting on a Circle, and Related Issues

Alan Hoenig

Owing to the generality of both *TEX* and *METAFONT*, it's easy to typeset in and on circles. Here's how.

#### The *METAFONT* Part

*TEX* can't actually turn characters on their side; we ask *METAFONT* to create special fonts where each character in the font is rotated around its reference point (the lower left corner of the bounding box of any character). Then *TEX* properly positions characters from the rotated fonts to achieve the illusion of circular typesetting. We need one rotated font for each position on the circle.

What does it mean to typeset characters around the circumference of a circle? I imagined a regular

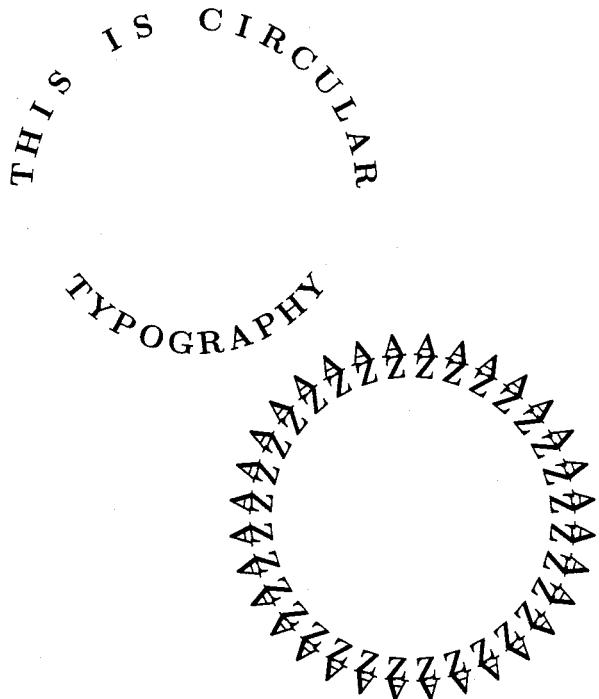


Figure 1. What this article is about.

polygon inscribed in the circle. The vertices of the polygon touch the circle from the inside, and the faces of the polygon form bases on which each character sits. Since each base is the same length as any other, I abandoned the concept of variable width typesetting on the circle; this accounts for the visually unsettling appearance of some circular typesetting. Later we will center each character on its base.

Let the bases be numbered from 0 to  $n - 1$ ; there are a total of  $n$  sides to this polygon. (It's more convenient to label the faces starting with 0 rather than 1.) Figure 2 shows a portion of such a circle with the first few faces. Notice that the zero-th face is at the "nine o'clock" position on the circle; that's because we read from left to right.

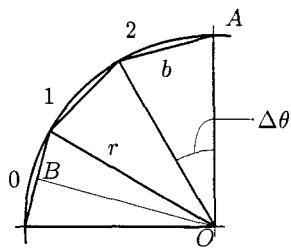


Figure 2. The inscribed  $n$ -gon on which we imagine placing rotated letters. The point  $B$  bisects its face.

For this article, I generated a sequence of rotated `cmbx12` fonts, and if we let  $b = 12$  pt, and imagine there to be room for 32 characters on the circumference of the circle, then the circle's radius must be 61.21 pt.

This follows from Figure 2 since

$$b/2 = r \sin(\Delta\theta/2).$$

If  $n$  is the number of faces in the inscribed polygon, then  $\Delta\theta = 2\pi/n$  or  $\Delta\theta/2 = \pi/n$ . Given  $n = 32$  (then  $\Delta\theta = 11.25^\circ$ ) and  $b = 12$  pt, we must have  $r \approx 61.21$  pt.

Recall the way METAFONT files are organized. Parameter files (such as `cmbx12.mf`) call driver files (such as `roman.mf`), which contain further details about the organization of the particular font. Finally, this driver calls several program files containing the instructions for generating the actual characters in the font. We will need to make changes to the parameter and driver files; the program files remain untouched.

I took the file `cmbx12.mf` and made 32 copies of it, named `cmbx1200` through `cmbx1231`. The idea is that file `cmbx12nn.mf` generates the font whose letters are rotated to stand on face  $nn$  of our

inscribed polygon. Make a copy of `roman.mf`, and call it `roroman.mf` (a rotated Roman font driver).

The changes to these files are essentially those which control the rotation of the font. The proper positioning of these characters involves knowledge of the trigonometric functions (sines and cosines) of certain angles. METAFONT does trig calculations very well, whereas `TeX` does them not at all. Therefore, we also need to pass the necessary trigonometric information to `TeX` for its use. We do this using the `fontdimen` mechanism.

Any font has several global characteristics that are helpful in typesetting. In a non-math font, these include things such as the width of a quad, the amount of stretchability of an interword space, and so on. These necessary quantities typically occupy positions `fontdimen1` through `fontdimen7`, but it's possible to create as many `fontdimen` parameters as needed. Note, for example, that if METAFONT stores a value of 1 (say) in `fontdimen10`, then `TeX` will read `\fontdimen10` for that font as 1 pt. `TeX` appends units of points to METAFONT's numerical `fontdimen` values.

### Changes to Files `CMBX12nn.MF`

The parameter files need few changes. At the beginning of each file, modify the comments to remind yourself of the changes you will have made. I also adjusted the value of the parameter `ligs = 0` to suppress ligatures. The last line of the file should be `generate roman;` change that to read

`generate roroman.`

The remaining changes are new lines which immediately precede this line, and they should look like this:

```
numeric wedge_angle;
wedge_angle=360/32;
numeric face; face=0;
numeric rotation_angle;
rotation_angle=
90-(face+.5)*wedge_angle;
fontdimen9: face, rotation_angle;
fontdimen11:
sind wedge_angle, cosd wedge_angle;
fontdimen13: % for r=61.21pt
sind(rotation_angle),
cosd(rotation_angle);
fontdimen15: % for r=30.61pt
sind(90-2(face+.5)*wedge_angle),
cosd(90-2(face+.5)*wedge_angle);
fontdimen17: % for r=15.30pt
sind(90-4(face+.5)*wedge_angle),
cosd(90-4(face+.5)*wedge_angle);
```

This puts various parameters in **fontdimens** nine through seventeen. The rotation angle is the angle by which we need to rotate a letter from the vertical so it will sit on its proper face on the underlying  $n$ -gon. The rotation is done in a counter-clockwise direction, as per the usual METAFONT convention. In figure 2, angle  $\overline{AOB}$  is the rotation angle for the letter that will sit on face 1. Notice that line  $OB$  bisects the wedge angle and is perpendicular to the face, which it bisects.

These lines should be the same in all of the rotated font parameter files, except for the line defining the value of **face**. In file **cmbx12nn**, the appropriate definition should be **face=nn**.

### Changes to **roroman.mf**

METAFONT can rotate the elements it draws as a matter of course, so we need only the following few alterations to **roroman.mf**.

```
currenttransform:=currenttransform
  rotated rotation_angle;
def t_=transformed
  currenttransform enddef;
```

These statements should appear immediately following the line

```
mode_setup; font_setup;
```

and in any case before the sequence of **input** statements that follows.

METAFONT's **currenttransform** applies a transform to all the pictures it generates. We simply define this transform to include a rotation by the current value of the rotation angle, and METAFONT does the rest.

### Thirty-Two New Fonts

Now, generate 32 new fonts. The METAFONT command line you need is

```
mf \&cm \mode=corona; input cmbx1200
```

and so on for the remaining 31 fonts. Minor variations will be necessary depending on your particular system. For example, you will need to select the proper **mode** name. In PCMETAFONT, for example, you conclude the line with the switches **/a=99/t**. Don't forget to change **input cmbx1200** to **input cmbx1201**, and so on. After creating each METAFONT font file, you need to transform the generic font file to a **pk** file via the utility **gftopk**; typically the command line to do that looks something like

```
gftopk cmbx1200.300 cmbx1200.pk
```

Finally, move the **tfm** file to wherever all your other **tfm** files are (probably in a directory named something like **\tex\textfms**) and move the **pk** files to their proper directory, something like **\tex\pixel\dpi300** for laser printer fonts; change the '300' to the resolution of your printer. (If your pixel files are organized according to the older convention involving numbers like 1500 and so on, the determination of where to place these fonts is less straightforward. In general, though, these font files should reside in the same region of your hard disk as do the fonts you use for normal 10 pt, **\magstep0** typesetting.)

I confess I only generated the uppercase letters to these rotated fonts to save my time and disk space. If you elect to follow suit, you'll have some minor additional changes to make to **roroman**—namely, comment out all but the first **input** statement in that file. You'll probably want to create batch files to generate your fonts, convert them to **pk** form, and move them to the proper directories.

### A **TEX** Digression

*As a warmup exercise*

we can do something simpler than circular typesetting. We will first typeset on an angle. To typeset up a 45-degree incline, we need a special font which I named **zcmr10**. I deviated from my naming scheme because no face is inclined at the proper angle when there are 32 faces in the polygon. In **zcmr10.mf**, let the rotation angle be 45 (degrees). Most of the **TEX** macros that are responsible for placing the letters properly appear somewhere in *The T<sub>E</sub>Xbook*; as is so often the case, doing something interesting with **TEX** is a matter of the artful extraction of the relevant bits and pieces from *The T<sub>E</sub>Xbook*.

The macros depend on a **\getfactor** macro. It takes a single argument, namely a particular **fontdimen** for a certain font, and returns the value of that **fontdimen** stripped of the units of points. This macro is largely adapted from an example in Appendix D (page 375). Watch closely.

```
{\catcode`p=12 \catcode`t=12
 \gdef\\#1pt{#1}}
 \let\getfactor=\\

```

Thus, if `\the\fontdimen1\tenit` is '0.25pt', then

```
\getfactor\the\fontdimen1\tenit
```

will yield 0.25 in some context where 0.25 makes sense.

We have to sidestep TeX's typesetting mechanism, since we are not setting characters on a common baseline, and we appropriate part of the solution to exercise 11.5, in which we learn how to seize individual tokens in a list. Here's the relevant code.

```
\def\dolist{\afterassignment
 \dodolist\let\next= }

\def\dodolist{\ifx\next\endlist
 \let\next\relax
 \else \\ \let\next\dolist \fi
 \next}
\def\endlist{\endlist}

\def\\{%
 next char letter or space?
 \expandafter\if\space\next\addspace
 \else\point\next\fi}
```

Macro `\addspace` (see below) is responsible for leaving spaces in the angle copy. The macro `\point`, drawn from Appendix D, is used to position the current character. In order to use these macros, we need to initialize certain registers and fonts.

```
\newdimen\x \newdimen\y
\def\initialize{\global\x=0pt
 \global\y=0pt }
```

We will depend on `\newcoords` to compute the coordinates for the reference point of the next character. We use analytic geometry to determine

$$\Delta x = -\sin \theta \cdot \text{wd0}$$

$$\Delta y = \cos \theta \cdot \text{wd0}$$

where  $\theta$  is the angle of inclination of the type from the vertical (here  $\theta = 45^\circ$ ) and `\wd0` is the width of the current character or space which is in `\box0`. Then,  $x \leftarrow x + \Delta x$  and  $y \leftarrow y + \Delta y$ .

```
\font\anglefont=zcmr10 % rotated font
\newdimen\DeltaX \newdimen\DeltaY
\def\newcoords{%
 \DeltaX=\expandafter\getfactor
 \the\fontdimen14\anglefont \wd0
 \DeltaY=\expandafter\getfactor
 \the\fontdimen13\anglefont \wd0
 \global\advance\x by-\DeltaX
 \global\advance\y by\DeltaY }
```

`\getfactor` strips the 'pt' from `\fontdimen13` and `\fontdimen14` and uses the resulting numbers—values of sine and cosine for an angle—as coefficients of the width of the box containing a space.

Here is the TeX code for `\addspace`, which determines how much space to leave between words.

```
\newbox\spacebox
\setbox\spacebox=\hbox{\ }
\def\addspace{\setbox0=
 \copy\spacebox \newcoords}
```

The `\point` macro that I use is slightly different from the one Donald Knuth provides in Appendix D. Here is its code.

```
\def\point#1{%
 \setbox0=\hbox{\anglefont #1}%
 % used by \newcoords
 \setbox2=\hbox{\anglefont #1}%
 % used for typesetting
 \wd2=0pt \ht2=0pt \dp2=0pt
 \rlap{\kern\x \raise\y \box2}%
 \newcoords}
```

Finally, the `\angletype` macro puts all the pieces together.

```
\def\angletype#1{\initialize
 \leavevmode\setbox0=
 \hbox{\dolist#1\endlist}%
 \box0 }
```

The instruction `\angletype{Angle of Repose}` was sufficient to typeset the subject of Figure 3.

Angle of Repose

Figure 3. Typesetting at an angle.

Angle typesetting might be useful when you prepare advertising copy, and when you need to typeset column headings on tables with very narrow columns, as in Figure 4.

### Typesetting on Circles

Once the angle-setting macros are in place, we need to alter details to accomplish typesetting on a circular path. On a circle, things change as we move along the circumference—we have to keep track of our position along the circumference, and at each new face we have to select the appropriate font.

The macros `\getfactor`, `\dolist`, `\dodolist`, and `\newcoords` remain the same. (In `\newdimen`, we rename

	Port	Claret	Burgundy	Champagne
1986	0	6	6	0
1985	0	7	7	7
1984	0	4	4	0
1983	7	6	7	6

Figure 4. A portion of a table with narrow columns. This is a portion of a table showing quality of recent vintages. The numbers give quality in a scale of 1 through 7; 0 means the wine is unrated.

\addspace to \newcoords.) The first new macro will determine the coordinates to the next vertex of the underlying polygon on which we place each type. We identify these coordinates as  $x_i$  and  $y_i$ . First we initialize the coordinates. The initial vertex  $(x_0, y_0)$  has coordinates  $(-r, 0)$ . \faceno is a numeric register containing the current face number (recall that we draw a correspondence between position along the circumference and a particular face of the inscribed 32-gon). Various radius-like quantities will later enable us to typeset around circles of varying radius.

```
\newcount\faceno
\newdimen\Bigradius \newdimen\Radius
\newdimen\radius \newdimen\r
\Bigradius=61.21pt
\Radius=.5\Bigradius
\radius=.5\Radius
\r=\Bigradius
\newdimen\x \newdimen\y
\def\initialize{\font\anglefont=
  cmbx1200 \global\faceno=0
  \x=-\r \y=0pt }
```

Given vertex  $(x_n, y_n)$ , we can get the next vertex (travelling clockwise) via

$$\begin{aligned} x_{n+1} &= x_n \cos \Delta\theta + y_n \sin \Delta\theta \\ y_{n+1} &= -x_n \sin \Delta\theta + y_n \cos \Delta\theta \end{aligned}$$

(see, e.g., David Salomon's article in *TUGboat* 10, no. 2, p. 213, July, 1989). We calculate these quantities using registers \dimen0, \dimen1, and \dimen2.

```
\def\nextpointt{%
  \dimen0=\expandafter\getfactor
    \the\fontdimen12\anglefont \x
  \dimen2=\expandafter\getfactor
    \the\fontdimen11\anglefont \y
  \advance\dimen0 by\dimen2
  \dimen1=\dimen0
```

```
\dimen0=-\expandafter\getfactor
  \the\fontdimen11\anglefont \x
\dimen2=\expandafter\getfactor
  \the\fontdimen12\anglefont \y
\advance\dimen0 by\dimen2
\global\x=\dimen1 \global\y=\dimen0}
\def\nextpoint{\nextpointt
  \preparefornextface}
\let\newcoords=\nextpoint

\newcount\lastface \lastface=31
\def\preparefornextface{%
  \global\advance\faceno by 1
  \ifnum\faceno>\lastface
    \global\faceno=0
    \message{There may be too many
      letters in your circular message!}%
  \else \ifnum\faceno<10
    \font\anglefont=cmbx120\the\faceno
  \else \font\anglefont=cmbx12\the\faceno
  \fi \fi}
```

Macro \preparefornextface changes fonts for the next face of the underlying polygon, and uses a numerical register \faceno for that purpose.

We won't use the coordinates  $(x_i, y_i)$  for typesetting, because that would put the reference point of the type at the vertex of our underlying, imaginary 32-gon. It is much better to center the type on its base. The centering macro \setonbase assumes that \box2 contains the current character and the corrected coordinates are  $(x'_i, y'_i)$ .

If  $w$  is the width of the type and  $b$  is the length of the base, then the vector  $\Delta r$

$$\Delta r = \frac{b-w}{2}(\cos \theta, \sin \theta)$$

provides the correction to  $r = (x_i, y_i)$  so that if we place the reference point of the type at the point  $r' = r + \Delta r$ , then it will be centered on that base.  $\theta$  is the rotation angle.

We can easily get  $\Delta r$  from  $r$  since the two vectors are perpendicular to each other. Given that  $r = r(-\sin \theta, \cos \theta)$ , then either of  $\pm(\cos \theta, \sin \theta)$  are perpendicular to it. Since  $\Delta r$  represents an offset in the clockwise direction, we choose the + sign.

```
\newdimen\xprime \newdimen\yprime
\def\setonbase{%
  \curr char in \box2
  \xprime=\x \yprime=\y
  \baseoffset=.5\base
  \advance\baseoffset by-.5\wd2
  \dimen0=\expandafter\getfactor
    \the\fontdimen14\anglefont \baseoffset
  \dimen2=\expandafter\getfactor
```

```
\the\fontdimen13\anglefont \baseoffset
\advance\xprime by\dimen0
\advance\yprime by\dimen2 }
```

Finally, we need a slightly altered `\point` macro, and `\circumtype` puts everything together.

```
\newdimen\base \base=12pt
\newdimen\baseoffset
\newtoks\currchar
\def\point#1{%
  \currchar=\expandafter{\#1}%
  \setbox2=\hbox{\anglefont
    \the\currchar}%
  \setonbase
  \wd2=0pt \ht2=0pt \dp2=0pt
  \rlap{%
    \kern\xprime \raise\yprime\box2}%
  \newcoords}

\def\circumtype#1{%
  \initialize
  \setbox0=\hbox{\dolist#1\endlist}%
  \leavevmode \box0 }
```

Figure 5 shows the alphabet around a circle. If the irregular rhythm of the type due to placing variable width type at equal intervals bothers you, you might want to consider using a monospaced font like `cmtt10` instead of the `cmbx12` that I used.



Figure 5. Circular typesetting.

### Smaller Circles

Because 32 is divisible by four, it is easy to typeset on circles that are one-half and one-quarter the radius of the original circle. Such cartouches would accommodate 16 and 8 characters around their circumferences. To do this right, we would need

additional trigonometric values in `cmbx12nn.MF` so that an enhanced version of `\setonbase` can compute  $\Delta r$  properly. That's why we included information for `fontdimens` 15 through 18 in the `METAFONT` parameter files.

Actually, the changes to `\setonbase` are extensive and I have not done them at this time. If you decide you want to, here are some things to keep in mind. When we shrink the radius, we need to increase the wedge angle. Halving the radius requires doubling the wedge angle (provided the length of the base remains constant), and so on. At a half radius, for example, we skip every other vertex of the original 32-gon. In Figure 6, we set a letter on faces  $AB$  and  $CD$  (closer to the center of the circle, though) while skipping faces  $BC$  and  $DE$ . However, we need `fontdimen` information from the skipped fonts to get information about the vectors  $\Delta r$ . In Figure 6,  $OB$  is perpendicular to  $AC$ . We need to invoke and save information from that skipped font.

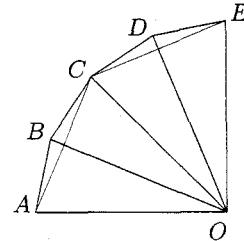


Figure 6. Typesetting when we change the radius.

However, it's easy to do "poor man's" typesetting around smaller circles if we adopt a "dummy" version of the `\setonbase` macro. Here's all we need do.

```
\r=\Radius
\def\newcoords{\nextpoint
  \nextpoint}
\def\setonbase{\% dummy def'n
  \xprime=\x \yprime=\y}
```

To typeset around the smallest circle, simply set

```
\r=\radius
```

and

```
\def\newcoords{\nextpoint \nextpoint
  \nextpoint \nextpoint }
```

Because of the do-nothing version of `\setonbase`, the reference point of each letter coincides with the

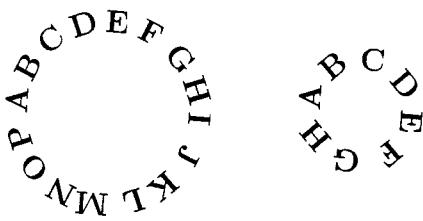


Figure 7. Typesetting around small circles.  
vertices of the underlying 16-gon or octagon. Somehow, though, at smaller radii, this is less visually unsettling than we would expect (see Figure 7).

### On the Inside of a Circle

Suppose we wanted to typeset around the *inside* of a circle. In light of the foregoing, one approach is to simply “METAFONT up” a new set of 32 fonts using a slightly different expression for the rotation angle, but it is possible to use the fonts we already have. For example, on face 1, we use font `cmbx1201` to determine type placement information, but we typeset the letter using the font that would normally appear diametrically opposite it (in this case, face 17). Given a face  $n$ , its opposite face  $n_{\text{opp}}$  must satisfy

$$|n - n_{\text{opp}}| = N/2$$

where  $N$  is the total number of faces and both  $n$  and  $n_{\text{opp}}$  must be non-negative integers less than  $N$ . (Remember,  $N = 32$  for our largest circle.)

When we use the font that belongs at the opposite face, we need to keep two points in mind. First of all, the reference point of the opposite font lies not at vertex  $n$ , but at vertex  $N + 1$ , the next clockwise vertex (think about it). We can take this into account in our initialization macro. When using opposing fonts, the initial position of the type on face 0 is not at  $(-r, 0)$  but at  $(-r \cos \Delta\theta, r \sin \Delta\theta)$ .

Because of the displacement of the reference point, the vector  $\Delta r$  that the `\setonbase` macro uses must point in the counterclockwise direction. These changes dictate the following new macros which contribute to the construction of macro `\circrintype`.

```
\newcount\oppface
\def\setinbase{%
  \xprime=\x \yprime=\y
  \baseoffset=.5\base
  \advance\baseoffset by-.5\wd2
  \dimen0=\expandafter\getfactor
  \the\fontdimen14\anglefont\baseoffset
  \dimen2=\expandafter\getfactor
```

```
\the\fontdimen13\anglefont\baseoffset
\advance\xprime by-\dimen0
\advance\yprime by-\dimen2
\oppface=\faceno \advance\oppface by0
\ifnum\faceno<16
  \advance\oppface by16
\else \advance\oppface by-16
\fi
\ifnum \oppface<10
  \font\oppfont=cmbx120\the\oppface
\else
  \font\oppfont=cmbx12\the\oppface
\fi \setbox2=
\hbox{\oppfont \the\currchar}

\def\ininitialize{\font\anglefont=
  cmbx1200 \global\faceno=0
  \x=-\expandafter\getfactor
  \the\fontdimen12\anglefont \r
  \y= \expandafter\getfactor
  \the\fontdimen11\anglefont \r }

\def\circrintype#1{\bgroup
  \let\setonbase=\setinbase
  \let\initialize=\ininitialize
  \initialize
  \setbox0=\hbox{\dolist#1\endlist}%
  \leavevmode\box0 \egroup}
```

Figure 8 shows what to expect from inscribed circular typesetting.



Figure 8. Typesetting inside a circle.

Incidentally, here are the commands I used to generate the top of Figure 1.

```
\circumtype{%
  THIS IS CIRCULAR~~~~~}%
\circrintype{%
  ~~~~~YHPARGOPYT~~~}%
```

I suffered plenty of setbacks *en route* to a working set of circular macros. Sometimes the results of faulty macros were interesting in their own right. Take a look at the best such mistake in Figure 9.

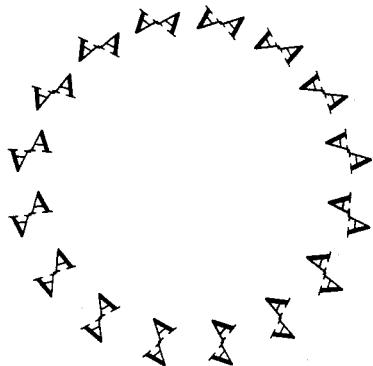


Figure 9. Mistake.

If you try this stuff yourself, note that circular typesetting may throw your previewer and device driver for a loop (apt?). You have been warned.

◊ Alan Hoenig  
17 Bay Avenue  
Huntington, NY 11743  
(516) 385-0736

## Graphics

### On the Implementation of Graphics into **TEX**

Gerhard Berendt

#### 1 Abstract

The problem of implementing more complex pictures than are provided by the **LATEX** picture environment into a typical PC version of **TEX** is discussed. In the first part of the article (Sections 2 and 3) a solution is presented which circumvents the usual limitation of the restricted main memory of **TEX** and respects the moderate hash size of the PC versions of **TEX**. This solution remains, however, totally within the frame of **TEX**. In the second part (Sections 4 to 7) a solution to the problem is given

which makes use of PostScript within the **TEX** environment.

#### 2 Introduction

While **TEX** is a very powerful tool for producing mathematical and technical texts, it has its well-known deficiencies as far as the implementation of graphics is concerned. The problem is twofold:

- The hash size of about 3000 for a typical PC version of **TEX** limits the complexity of macro packages which implement graphics. It is, e.g., impossible to add the rather comfortable **PICTEX** macro<sup>1</sup> package to **LATEX** because of an overflow of the hash size. In order not to surpass the given hash size, it is therefore necessary to use a more moderate graphics macro package, if the **LATEX** environment is obligatory. Our solution to this problem will be presented in the next section.
- Another more subtle problem results from the fact that even a picture of only moderate complexity—if it is not produced by characters of special fonts (as is the philosophy in **LATEX**)—might overflow the main memory of **TEX**. It is then impossible to compile a page which contains this picture. The only way out of this difficulty is to compile text and picture separately and either to combine the two **dvi** files afterwards or to print text and picture in two runs.

In the first part of this article, we present a compromise solution to both problems which:

- enables the user to produce texts plus included pictures of moderate complexity; and
- needs nothing but **LATEX** running on a PC together with a small graphics macro package, a parameter file extraction program and (optionally) another utility program which automates the creation of the picture input.

Our solution relies neither on special output devices or files (e.g. laser printers or PostScript files) nor on drawing programs or special picture formats. Instead, the pictures are drawn within the **LATEX** picture environment which is enriched by a few graphics macros from the extended **epic** style.

---

<sup>1</sup> M.J. Wichura, *TUGboat* 9, no. (2), p. 193, 1988

### 3 Graphics via L<sup>A</sup>T<sub>E</sub>X techniques

In the following, we use the rather small Enhanced Picture Environment package,<sup>2</sup> which is given by the `epic` style format, together with the `bezier` style format<sup>3</sup> for drawing curves, and finally a few additional macros for producing gray tones within certain parts of the picture. As the `bezier` style and the `epic` style are public domain files, we concentrate on the additional macros, which are included in the file `neubild.sty`.<sup>4</sup>

#### 3.1 The picture input

There are two main macros within this style file, `\varbild` and `\VARBILD`. The macro `\varbild` has 7 parameters, 3 of which are optional; the complete macro goes as follows:

```
\varbild[#1](#2,#3)(#4,#5)#6#7
```

#1 is optional; its possible values are `<l>`, `<r>`, `<c>` or `<v>`, the default being `<v>`.

The values of #1 have the following meanings:

`<l>`: The picture is located at the left margin of the text, and the text flows around the picture on its right side.

`<r>`: The picture is located at the right margin of the text, and the text flows around the picture on its left side.

`<c>`: The picture is centered, and there is no text neither at the right nor at the left of the picture.

`<v>`: `<v>=<r>` if the pagenumber is odd, else `<v>=<l>`.

#2 and #3 denote the width and the height of the picture in multiples of `\unitlength`. The default `\unitlength` provided by `neubild.sty` is 1 mm.

#4 and #5 are again optional; the defaults are `#4=#5=0`. They denote the origin's translation in the coordinate system of the L<sup>A</sup>T<sub>E</sub>X picture environment, again as multiples of `\unitlength`.

#6 denotes the contents of the picture environment (without the control sequences

<sup>2</sup> Copyright (C) Sunil Podar, Dept. of Computer Science, SUNY at Stony Brook, NY 11794, U.S.A.

<sup>3</sup> Copyright (C) 1985 by Leslie Lamport

<sup>4</sup> As the macro package was developed in the context of a German book project, the names of the macros are given in German, which we hope will not give rise to irritations.

`\begin{picture}` and `\end{picture}`).

#7 finally is a correction parameter; it denotes the number of lines by which TeX's calculation for the picture depth should be shortened. #7 can be positive, zero or negative and does not have any meaning with the option `#1=<c>` (but must be set to zero in this case).

The macro `\VARBILD` has 8 parameters, 3 of which are optional. Here, instead of including `picture` commands explicitly as with argument #6 above, we pass filenames so that the picture may be laid out on a separate run of TeX. Two filenames are needed for each picture: one for a L<sup>A</sup>T<sub>E</sub>X 'driver' file which places the picture in the proper position on a page, the other for a file containing `picture` commands.

The complete macro goes as follows:

```
\VARBILD[#1](#2,#3)(#4,#5)#6#7#8
```

The parameters #1 through #5 have the same meaning as in the macro `\varbild`; again, #1, #4 and #5 are optional.

#6 has the same meaning as the parameter #7 in `\varbild`.

#7 is a string in DOS-format; it denotes the name (without extension) of a *L<sup>A</sup>T<sub>E</sub>X driver* file, which becomes the source file of the picture in question on a separate run of TeX (since `\VARBILD` only provides an empty frame of the correct picture dimensions within the text file).

#8 finally — again a string like #7 — denotes the name of a *picture source* file which will contain the `picture` commands called by the corresponding *L<sup>A</sup>T<sub>E</sub>X driver*.

The macro `\varbild` is used whenever all the commands of a picture shall be enclosed within the text file, while the macro `\VARBILD` creates an empty frame of the chosen dimensions within the text file and, in addition, writes to a parameter file the information needed to build pure picture files, each of which will be placed at the correct position on its respective page. If the `\VARBILD` alternative is chosen, the text file must start with the line

```
\immediate\openout\bilder = <parameter file>
```

and its last line must be

```
\closeout\bilder
```

After compilation of the main text file, which will also generate the parameter file, the picture files

have to be created with the program MAKEPIC via the command line

MAKEPIC *<parameter file>*

This program produces for each line of the parameter file (*i.e.* for each occurrence of \VARBILD in the source file) a *<L<sup>A</sup>T<sub>E</sub>X driver>* file with the name given by argument #7 of \VARBILD. This file contains nothing but the picture environment at the correct place.<sup>5</sup> It can therefore be compiled as usual and be printed onto the appropriate page separately. If the placement of the picture is not quite accurate, it is possible to move the picture by applying the program MAKEPIC once more to the parameter file, this time using the option of moving the coordinate system in question.

### 3.2 Picture creation and implementation

While it is rather easy to construct pictures within the picture environment of L<sup>A</sup>T<sub>E</sub>X as long as there are only lines, vectors, circles and text involved, it is very cumbersome to introduce picture input which contains Bézier functions or other complex features. Therefore, the MAKEPIC program creates a L<sup>A</sup>T<sub>E</sub>X file with only the frame of the picture, while the actual picture input is contained in an input file, the name of which is given by the parameter #8 in the \VARBILD macro. The whole picture input—apart from the \begin{picture} and \end{picture} lines—is thus expected as the contents of a *<picture source>* file. This file may be created manually or via the additional utility program PICTPLUS. This program asks either for a function  $x = x(t)$ ,  $y = y(t)$  or for an area to be shadowed, and will in return produce a L<sup>A</sup>T<sub>E</sub>X input file, which can be inserted into the *<L<sup>A</sup>T<sub>E</sub>X driver>* file in place of the line \input *<picture source>* or can be left within the working directory as the *<picture source>* file itself.

Thus, the whole procedure for including a picture of moderate complexity into a L<sup>A</sup>T<sub>E</sub>X text file by help of the \VARBILD goes like this:

1. Insert the line

`\immediate\openout\bilder=<parameter file>`

at the very beginning, and the line

`\closeout\bilder`

as the last line into your L<sup>A</sup>T<sub>E</sub>X source file.

2. Introduce the option neubild into your document style.

---

<sup>5</sup> It is, of course, also possible to write the picture source file by hand, using the information from the parameter file.

3. Write the source file and insert a \VARBILD macro at the beginning of each paragraph where a picture should be placed. Be sure that the text that will flow around the picture frame does not contain any \par macro (this does not apply if the option #1=<c> is chosen). It is advisable to write the whole paragraph which surrounds a picture in \sloppy mode, since the \textwidth is reduced within this paragraph whenever you produce a picture which does not cover the whole width of the page.
4. Compile the source file and thereby create the parameter file automatically because of step 1.
5. Run the program MAKEPIC on the parameter file to create the *<L<sup>A</sup>T<sub>E</sub>X driver>* files for each occurrence of a \VARBILD.
6. Create the *<picture source>* files for each of the *<L<sup>A</sup>T<sub>E</sub>X driver>* files either manually or via the program PICTPLUS. Insert these files into the *<L<sup>A</sup>T<sub>E</sub>X driver>* files in exchange for the line \input *<picture source>* or put these \input files into your working directory.
7. Compile the *<L<sup>A</sup>T<sub>E</sub>X driver>* files separately.
8. Print the main dvi file first and then the *<L<sup>A</sup>T<sub>E</sub>X driver>* dvi files separately onto those printer output pages where the corresponding empty frames of the pictures have been produced. If for a certain picture there are slight deviations from the correct placement, compile the *<L<sup>A</sup>T<sub>E</sub>X driver>* file in question once again, this time using the option of moving the coordinate system appropriately, or use the facilities of the MAKEPIC program to shift the picture slightly.

While the whole procedure might look a little bit complex, one should bear in mind that—apart from the two auxiliary programs MAKEPIC and PICTPLUS—there are no requirements necessary in addition to the pure T<sub>E</sub>X mechanism. In the second part of this paper a more elegant solution to the problem, using PostScript files, will be presented.

### 4 Implementation of graphics via PostScript

With the advent of software driven PostScript interpreters like FREEDOM OF PRESS it has become reasonable to print T<sub>E</sub>X files via PostScript on a multitude of cheap printers. Of course, printing time is much longer in PostScript than it is with a T<sub>E</sub>X driver like PC DOT or PTI JET. It is therefore worthwhile to switch to the PostScript scheme only if the results are inconvenient or insufficient otherwise. This is the case if even only moderately

complex pictures are to be included into **TEX** or **LATEX** text files, because on the one hand there exists only a very limited variety of picture elements within **TEX** or **LATEX**, and — which is much worse — on the other hand the compilation of the source file may fail due to memory restrictions of **TEX**.<sup>6</sup> Therefore, if time is not an important factor it might be convenient to create a **dvi** file from the source text together with a PostScript file for each of the pictures to be included and then to combine these files into one PostScript file, for instance, via the PTIPS driver. PTIPS enables the user to insert any ordinary PostScript file at an arbitrary position into the main file by help of the **TEX** \special command. In the following, a scheme is developed to use this feature in **LATEX** files which contain pictures of moderate complexity.

### 5 Preparation of the main text file

In order to prepare the main text file for the inclusion of pictures, a method similar to the one given in the first part of this paper is used. Again, the option **neubild** has to be added to the documentstyle of the main file, and a parameter file has to be opened by the line

```
\immediate\openout\bilder=<parameter file>
```

and closed by the line

```
\closeout\bilder.
```

The file **neubild.sty** contains, in addition to the macros already mentioned, the macro \varpsbild which writes an entry into the parameter file and sets an empty frame of the desired size and position within the main file. In addition to this, \varpsbild puts a mark equivalent to the line

```
\special{ps:bildname.ps}
```

at the correct position in the main text **dvi** file after compilation. This mark enables the PTIPS driver to insert the PostScript file **bildname.ps** at this position as it creates the PostScript file of the main text.

The macro \varpsbild is given as

```
\varpsbild[#1] (#2,#3) (#4,#5) #6#7
```

where the parameters #1 to #5 have the same meaning as the corresponding parameters within the macro \VARBILD, while the parameters #6 and #7 have the following meaning:

#6 is mandatory; it is the name of the PostScript file which contains the picture elements to be put into the empty frame.

---

<sup>6</sup> Section 1.

#7 is mandatory; it has the same meaning as in the macro \varbild (it denotes the number of baselines by which **TEX** must underestimate the surrounding text to place the picture properly).

### 6 Preparation of pictures to be included

In principle, any PostScript picture file can be created manually by help of any appropriate editor using the PostScript language. Anybody who has a fluent knowledge of the PostScript language will certainly prefer this way to other possibilities. Since, however, many users of **TEX** are not so experienced in PostScript, we developed a small PASCAL program which can be used interactively to input the most common elements of the **LATEX** picture environment and output a complete PostScript file ready for insertion into the main text PostScript file. This program, **TEX2PS**, is menu-driven and allows construction of objects **<line>**, **<vector>**, **<curve>**, **<ellipse>** and **<text>** in any desired combination, solid or dashed, with or without shading. Moreover, multiple constructs of those objects can be performed in any order and PostScript program lines can be added at will manually. Thus, the elements of any picture which can be created within a slightly extended picture environment of **LATEX** (as, e.g., in the **epic** style) can be input to **TEX2PS** and be transferred to the corresponding PostScript description. By means of the procedure described above it is then possible to produce a PostScript file which combines the main text with any number of pictures of that sort. The combined file can be printed either directly by a PostScript printer or via a software interpreter like **FREEDOM OF PRESS** or similar program on many non-PostScript printers.

A few remarks should be made concerning the program **TEX2PS**. Though it is meant to be used without referring to the PostScript language, a slight knowledge of the PostScript graphic elements is recommended (of course, you might get the scheme by trial and error after a while). The assignments '*Draw*' (= 'stroke'), '*Fill*' and '*Eofill*' have their origin in the PostScript language and do exactly what they would do in a PostScript setting<sup>7</sup>; however, they are enclosed between '*gsave*' and '*grestore*' lines. It is therefore possible to shade an area first and then to draw its border lines without repeating the whole pattern. The option '*Link*' is not a PostScript operator; it is used to construct a continuous path of equal or different elements (for instance to be shaded afterwards).

---

<sup>7</sup> e.g. *The PostScript Language Reference Manual* by Adobe Systems Inc., Addison-Wesley, 1985

Finally, it should once more be emphasized that—if you have a good knowledge of the PostScript language—it is generally much more efficient to create the picture PostScript file by directly editing the picture rather than using the automated but necessarily clumsy version which is provided by the program TEX2PS.

## 7 The picture implementation

The method described above yields the following steps of procedure:

1. Start your main L<sup>A</sup>T<sub>E</sub>X text file with the line

```
\immediate\openout\bilder=<name of
parameter file>
```

and close it by the line

```
\closeout\bilder.
```

2. Introduce the option neubild into your document style.

3. Write your text file and introduce the line

```
\varpsbild...
```

with the appropriate parameters as explained above at any place where you want to insert a picture.

4. Compile the text file to the corresponding dvi file. This also produces the parameter file.
5. Create all the pictures in a PostScript setting either manually or by help of the program TEX2PS, taking into account the correct size and name of every picture (the parameter file contains these parameters for each picture involved).
6. Convert the main dvi file by help of the PTIPS driver, using all of the PostScript picture files, to the final PostScript file.
7. Print the final PostScript file either on a PostScript printer or via a soft interpreter like FREEDOM OF PRESS.

## 8 Conclusion

The two procedures described above are certainly not the most elegant ones for implementing graphics in T<sub>E</sub>X. As has been shown, the first method, however, has the advantage of not using any graphics input apart from that which is admissible in the L<sup>A</sup>T<sub>E</sub>X picture environment, and it is completely driver-independent. A somewhat similar approach to this problem is, for instance, given by M. Ballantyne and collaborators<sup>8</sup>; their method, however, is

---

<sup>8</sup> M. Ballantyne et al., TUGboat 10, no. 2, p. 164, 1989

at the moment not applicable within a L<sup>A</sup>T<sub>E</sub>X environment and, moreover, does not seem to work very well if the pictures are to be surrounded by text passages. On the other hand, that method can also be used to include complex tables into a T<sub>E</sub>X file.

We have not discussed the various methods which use graphics input from different drawing programs to be included into T<sub>E</sub>X source files. These methods depend heavily on the output format of the drawing programs (e.g. whether or not they are pixel oriented) as well as the ability of T<sub>E</sub>X drivers to implement the different graphic formats (usually by means of \special commands).

The second procedure allows the insertion of much more complex pictures into L<sup>A</sup>T<sub>E</sub>X text files at the price of using part of the PostScript machinery. We feel that it might be a good compromise if the time factor does not have first priority and the pictures to be inserted into the text are of moderate complexity.

A diskette containing the files used in these approaches can be ordered from the author. Please, enclose an empty diskette and DM 5,- for postage.

◊ Gerhard Berendt  
Institut für Mathematik I  
Freie Universität Berlin  
Arnimallee 2-6  
1000 Berlin 33  
Germany  
*berendt@fubinf.uucp*

---

## Including Macintosh Graphics in L<sup>A</sup>T<sub>E</sub>X Documents

Len Schwer

### Abstract

The basics of including Macintosh graphics in L<sup>A</sup>T<sub>E</sub>X documents are discussed for the person who is inexperienced at doing so. Because there is no universal way to incorporate such graphics, other than with scissors and glue, this article tries to be as general as possible, but ultimately references specific software and hardware, e.g. ArborText's DVIPS, Trevor Darrell's psfig macros, and an Apple LaserWriter+. The reader is assumed to have some knowledge of

the Macintosh interface, PostScript programming, and *LATEX* document preparation.

## 1 Background

The method of including Macintosh graphics in *LATEX* documents is very simple:

- create the graphic with the user's favorite Macintosh application;
- convert the graphic into its PostScript representation;
- transfer the PostScript file to the *LATEX* host machine;
- include the PostScript file in the *LATEX* document via the DVI-PostScript driver's `\special` command.

While this sounds like a relatively straightforward procedure, it gets complicated, sometimes very complicated. The complications arise from three elements:

1. There are several DVI-PostScript drivers available and they all treat the `\special` command differently. Many of the differences are simply syntactical, but some are more subtle and involve differences in the PostScript prologue which precedes the material from the *LATEX* document. The good news here is that there is an organized movement within the *TEX* community to develop standards for DVI drivers [1] and someday users may benefit from these standardization efforts.
2. Not all PostScript devices are the same. Macintosh QuickDraw, a PostScript language shorthand, in combination with various PostScript implementations of DVI drivers produces different results on different PostScript devices. For example, the ArborText DVI driver, DVIPS, and LaserWriter+ apparently cannot be coaxed into including Macintosh figures according to ArborText's instructions, while the same DVIPS generated file works flawlessly with an NEC PostScript printer.
3. Lastly, but of equal importance, the individuals who decide to dabble in this topic need a working knowledge of the Macintosh interface, PostScript programming, and *TEX* or *LATEX* document preparation.

The original *TUGboat* article on this topic by Hal Varian and Jim Sterken [2] appeared in March 1986. Since that time, there has been a dramatic increase in the number<sup>1</sup> of users of both *LATEX* and the Macintosh. New and 'old' *LATEX* users are becoming

Macintosh users for many reasons, not the least of which is the ability to easily produce high quality graphics for inclusion in their high quality typeset documents. Many of these new users are seeking ways to include Macintosh graphics in *LATEX* documents by means other than scissors and glue.

This article updates the information presented in the original *TUGboat* article and complements the information provided in a more recent article by J.T. Renfrow [3]. The present article is intended to serve as a general overview for the person who is new to including Macintosh graphics in *LATEX* documents. It tries to be general, when possible, by indicating how things are supposed to work, but in many places it is very specific. Where appropriate, mention will be made of available public domain software and possible sources for obtaining it.

There are two major sections in this article:

**Macintosh graphics.** This section describes capture or conversion of Macintosh graphics to an equivalent PostScript representation and changes needed for the *LaserPrep* file.

## Using BBFIG and psfig to include graphics.

This section describes a very useful set of macros that simplify inclusion of Macintosh graphics in *LATEX* documents.

## 2 Macintosh Graphics

The Macintosh was designed to be used with an Apple LaserWriter printer, a PostScript device. Thus all Macintosh applications need to support PostScript if they are to allow printing. We would like to capture the PostScript representation of a graphic in a file, transfer the file to the host machine where *LATEX* is used, and include the PostScript graphic file in a *LATEX* document. This section describes a 'universal' method for capturing the PostScript representation of a graphic in a file.

### 2.1 Capturing Macintosh graphics in a PostScript file

Unfortunately, few Macintosh applications, other than Cricket Draw and Adobe Illustrator, provide a menu option for generating a PostScript file. This is probably due to the existence of a universal technique for capturing any printer-directed application output into a PostScript file. The technique is quite simple:

After completing the graphic, select the *Print* option under the *File* menu. This produces

---

such as `comp.text.tex`, `comp.lang.postscript`, and `comp.sys.mac` on Usenet and *TeXhax* on the Internet or Bitnet.

<sup>1</sup> The growing number of these users is evident to those who monitor electronic information groups

the 'Print Dialog Box' which allows the user to select various options before sending a graphic to the printer. In the upper right-hand corner of the 'Print Dialog Box' is an OK button. Click the mouse down, but **do not release**, on the OK button. With the mouse still clicked down, depress and hold down the F key and then the Command (Apple or Flower) key (this combined key stroke is usually referred to as *Command-F*), then release the mouse button. A dialog box should appear stating that a PostScript file is now being created. Note: In *most* applications the Command key need not be depressed.

Recently, a very handy Macintosh application named **myPageSetup** by D.G. Gilbert has appeared<sup>2</sup> which activates a previously hidden selection box in the Print Dialog. The activated box is called **Disk File** and selecting it will cause the LaserWriter application to create a PostScript file rather than sending the file to the printer when the OK button is clicked. This eliminates the need for the somewhat clumsy Command-F keystroke sequence just described.

The above techniques will cause a file named 'PostScript0' (or, more generally, 'PostScriptn' where *n* is incremented by one for each PostScript file generated) to be created in one of several places:

- the folder where the generating application resides, *e.g.* where you keep MacDraw;
- the folder where the graphic was *launched*, *e.g.* where you have stored the graphic file;
- the DeskTop level of your startup disk;
- the System folder.

The exact location is application dependent<sup>3</sup>, but the Macintosh **FINDER** may be used to locate these PostScript files in any case.

Although these files are labeled as PostScript files, the files generated using the Command-F technique are not quite PostScript files, but are more correctly referred to as *QuickDraw* files. QuickDraw

<sup>2</sup> This freeware application is available via anonymous FTP from **sumex-aim.stanford.edu** in the directory **/info-mac/util** and probably from many other such Macintosh archives.

<sup>3</sup> An init named **LaserFix** by David P. Sumner modifies the print dialog box in the same manner as **myPageSetup**, but invokes a standard file location dialog box after the OK button is clicked. This allows a user to specify a folder name where all such PostScript files will be created. This init is also available from **sumex-aim.stanford.edu** in the directory **/info-mac/init**.

is a special PostScript shorthand created by Apple Computer.<sup>4</sup> NOTE: QuickDraw files will not produce a graphic image when sent to a PostScript printer unless a special initialization file has previously been sent to the printer.

The QuickDraw initialization file is commonly called a *LaserPrep* file and various versions of it are known as AppleDict Version #nn, where nn is the version number, *e.g.* AppleDict Version #70 (a.k.a. LaserPrep 70). The LaserPrep is a dictionary that translates QuickDraw into PostScript. The resulting translation does produce a printable graphic image on PostScript printers. Usually the LaserPrep file is downloaded only once to a PostScript printer connected to a Macintosh. This PostScript dictionary, called 'md' (for Macintosh Dictionary?), remains resident in the printer's volatile memory until the printer is powered down. A copy of the Macintosh's current LaserPrep file may be generated by following the Command-F procedure described above, but substituting *Command-K* before releasing the mouse clicked down on the OK button in the 'Print Dialog Box'. Using the Command-K key sequence generates a file named 'PostScriptn' that contains both the LaserPrep file and the contents generated by the Command-F key sequence; *i.e.* the LaserPrep file is inserted as a prologue to the QuickDraw file. The boundary between the LaserPrep and QuickDraw is located at the first occurrence of the string %%EOF, which is the last line of the LaserPrep file.

One more note about the LaserPrep file: near the bottom of the LaserPrep file are lines of hexadecimal numbers followed by several lines of zeros. The lines of hexadecimal numbers are very long and will break most file transfer programs. These lines may be shortened by inserting carriage returns at appropriate distances along the string. Alternatively, for the less faint-of-heart, these lines may be deleted from the LaserPrep file. More specifically, the lines between and including:

```
currentfile ok userdict/ ...
...
cleartomark
```

may be deleted. According to Bill Woodruff [4]

The dictionary [LaserPrep] includes some special encrypted assembly-language procedures that are proprietary to Apple Computer. If you check the "Faster BitMap Printing" option in the generic Macintosh Page Setup dialog, for example, you activate an Apple bitmap smoothing routine that will

<sup>4</sup> Some very interesting comments on the development of QuickDraw are provided in a brief article by Bill Woodruff [4].

not work if the installation of 'md' recognizes the printer is not from Apple.

## 2.2 Modifying the LaserPrep File

Woodruff [4] also comments:

LaserPrep was patched and fixed and extended and patched and fixed until it has reached its current state where even within Apple it is considered an embarrassing morass. Yet, it remains the world's most used PostScript program and the fundamental bottleneck through which almost all Macintosh printing is done. But to change it, even slightly, is to move a gigantic tectonic plate on which the entire superstructure of civilized Macintosh printing hangs in fragile balance.

In order to include Macintosh PostScript files generated with the Command-F technique described above with any DVI to PostScript driver, the LaserPrep file must be modified. There are two major types of changes:

1. Changes to keep the LaserPrep from altering the printer's status, *i.e.* changes to `statusdict`.
2. A change to prevent the included PostScript file from issuing a `showpage` or `copypage` command.

Listings of the modified Macintosh LaserPrep files, even 'differences' listings, are too lengthy for this article. Suitably modified versions of Macintosh LaserPrep #65 and #68 files<sup>5</sup> are available via anonymous FTP from `ymir.claremont.edu`. The modified version of LaserPrep #65 is derived mostly from the instructions issued by ArborText for their DVIPS driver.

The modified version of LaserPrep #68 was created by Trevor Darrell who claims that it should be compatible with Tony Li's public domain PostScript device driver DVI2PS when used with the appropriate `TEX` prologue. DVI2PS, the `TEX` prologue file, modified LaserPrep #68 file, and associated information are available via anonymous FTP from `linc.cis.upenn.edu` [130.91.6.8] in files `dvi2ps.tar.Z` and `lprep68.tar` of the subdirectory `dist/psfig`. These files are in Unix TAR format and, in the case of the `dvi2ps.tar.Z` file, Unix TAR and compressed binary format. However, after uncompressing the `dvi2ps.tar.Z` file and extracting the files (unTARing), you will find it also contains modifications for implementation on VMS systems; the unTARed `lprep68.tar` file con-

tains plain text files. Another source for these files is the recent Digital Equipment Corporation Users Society (DECUS) `TEX/LATEX` tape collection, prepared by Ted Nieland; for availability call DECUS at (505)480-3418 or contact your Local User's Group (LUG).

The modified LaserPrep file should be given an appropriate name, *e.g.* `LaserPrep68.ps`, and placed in a directory where DVIPS or DVI2PS can find it to include in DVI files. A suggested directory location is one of the directories searched by the logical `TEX$Inputs`. Also, the modified LaserPrep file must be included in `LATEX` documents before the occurrence of the first Macintosh PostScript file. The LaserPrep file is usually included prior to the `\begin{document}` statement *e.g.* by using ArborText's `\special` command,

```
\special{ps: plotfile LaserPrep68.ps global}
```

It is important to use the modified version of the LaserPrep file that corresponds to the LaserPrep used to generate the Macintosh graphic with the Command-F procedure. To determine which version is appropriate, search for a line like the following near the top of the file:

```
%INCLUDEProcSet: "(AppleDict md-figure)" 68 0
```

The number 68 in this example indicates that modified LaserPrep #68 should be used.

To determine the current version number of the LaserPrep file on your Macintosh, either (1) select the LaserPrep icon in the System folder and use `Get Info` under the `File` menu, or (2) when attempting to print a document from the Macintosh, look just to the left of the 'OK' button in the Print Dialog Box. In either case the number should be 4.0 or 5.2. 4.0 is equivalent to LaserPrep #65 and 5.2 is equivalent to LaserPrep #68. Basically, Macintosh system software 5.x, LaserPrep Version 4.0, and 'md' Version 65 all go together, as do Macintosh system software 6.x, LaserPrep Version 5.x, and 'md' Version 68. Simple, huh? This means that, technically, LaserPrep files such as LaserPrep #68 are referring to the version of the Apple dictionary and not the actual LaserPrep version number. Hopefully, someday soon, Apple will get all these numbers in sync.

The most recent version of the Macintosh LaserPrep file is #70, which is associated with LaserWriter version 6.0 and accompanies the latest release of the Macintosh operating system. LaserPrep #70 is considerably different from previous versions in that it contains information for Apple's implementation of Color QuickDraw. Requests to various electronic forums, and Trevor Darrell, for a suitably modified LaserPrep #70 indicate that knowledgeable LaserPrep hackers have not modified version #70. An ef-

---

<sup>5</sup> The hexadecimal strings at the bottom of the LaserPrep files have been deleted.

fective work-around is to install LaserPrep #68, *viz.* LaserWriter 5.2, on your Macintosh and rename it, for example to **LaserWriter68**. This version of the LaserWriter can then be selected via the *Chooser* before creating PostScript files.

### 3 Using BBFIG and psfig to Include Macintosh Graphics

**Acknowledgement:** BBFIG was authored by Ned Batchelder. **psfig** was developed and placed in the public domain<sup>6</sup> by Trevor J. Darrell. This subsection borrows quite liberally from Mr. Darrell's very nice documentation, *Incorporating PostScript and Macintosh Figures in TEX*. Current versions of the software and documentation are available via anonymous FTP from `linc.cis.upenn.edu` [130.91.6.8] in the sub-directory `dist/psfig`. This author, and undoubtedly many other **psfig** users, are indebted to Mr. Darrell for his outstanding programming skills and his unselfish willingness to share this software and knowledge with others. The author hopes that this document continues Mr. Darrell's spirit of freely sharing knowledge.

**psfig** is a TEX macro package that facilitates the inclusion of arbitrary PostScript figures in LATEX documents. The real advantage of using the **psfig** macros is that they work within the Adobe constructs for Encapsulated PostScript Files (EPSF) [5]. For the **psfig** user, that means that graphics can both be easily placed within a LATEX document and be printed on their own directly to a PostScript device; other systems for including PostScript graphics require a `currentpoint` to be set and the `showpage` to be explicitly disabled, thus disabling direct printing of the file.

To properly locate a PostScript figure, the DVI driver must know the size of the figure and its relative position on the printed page. This information is implicitly available within the various graphic constructs used in the PostScript file. The information *should* also be available explicitly in the Bounding-Box comment [6] of the PostScript file's prologue.

The bounding box encloses all the marks made on a page as a result of executing (printing) a Post-

---

<sup>6</sup> Copyright notice from the **psfig** source: "All software, documentation, and related files in this distribution of **psfig/tex** are Copyright (©) 1987 Trevor J. Darrell. Permission is granted for use and non-profit distribution of **psfig/tex** providing that this notice be clearly maintained, but the right to distribute any portion of **psfig/tex** for profit or as part of any commercial product is specifically reserved for the author."

Script program. The BoundingBox comment has four parameters:

**%%BoundingBox:**  $ll_x\ ll_y\ ur_x\ ur_y$

The four integer parameters, in units of points, represent the coordinates of the lower left ( $ll_x, ll_y$ ) and upper right ( $ur_x, ur_y$ ) corners of the bounding box in the default user (creator) coordinate system.

Although good PostScript programming practice dictates that the BoundingBox comment be provided in PostScript files, few Macintosh programs provide this information. Either the BoundingBox parameters are not specified, *e.g.* as with

**%%BoundingBox:** ? ? ? ?

which is the case for all Command-F generated PostScript files, or an entire  $8.5 \times 11$  page is specified, *e.g.* with

**%%BoundingBox:** 0 0 612 792

To supply the proper bounding box information, one can measure the required dimensions or use the PostScript utility BBFIG, which calculates the bounding box parameters from the graphic information implicit in the PostScript file. The BBFIG PostScript file is prepended to the PostScript file<sup>7</sup> for which the bounding box is to be determined, and the combined file is sent to a printer. The result is a printed image of the PostScript graphic with the bounding box drawn around the graphic and bounding box parameters listed below the bounding box; the bounding box parameters are also returned via the print job's log file, if the log file feature is implemented for the host machine's print queue.

The accuracy of BBFIG in determining the bounding box information is not as good as may be needed in some circumstances. If you notice your figures missing parts or wandering around the page, check the bounding box information. Of course, the exact bounding box information can be obtained by simply printing the figure and using a ruler, in points if possible, to measure the four coordinates. Measuring the bounding box coordinates is also necessary for graphics from some Macintosh applications such as Cricket Software's Cricket Graph. For some unknown reason, Cricket Graph figures contain an invisible (un-stroked) path around the entire edge of the paper.

Once the proper bounding box information has been added to the BoundingBox comment of a PostScript file, the file may easily be included in a LATEX document by using the TEX macro `\psfig`. Simply

---

<sup>7</sup> If the PostScript file was generated using Command-F (QuickDraw), then a *non-modified* LaserPrep must also be prepended to the QuickDraw file.

load the `psfig` macros at the beginning of your document with

```
\input{psfig}
```

then invoke the macro

```
\psfig{figure={input}}
```

where `input` is the name of a PostScript file. `psfig` will automatically position the figure at the current place on the page, and reserve the proper amount of space in L<sup>A</sup>T<sub>E</sub>X so that it does not conflict with any other objects.

For example, if we have a file called `piechart.ps` that contains the PostScript code to draw a pie chart, we would use the command

```
\centerline{\psfig{figure=piechart.ps}}
```

Since no mention of size is made in the above example, `psfig` would draw the figure at its natural size (as if it were printed directly by a PostScript printer.) If the pie's natural size is several inches across, which is a little large, the pie could be reduced with:

```
\centerline{%
```

```
\psfig{figure=piechart.ps,height=1.5in}}
```

The `height` option specifies how tall the figure should be on the page. Since no `width` is specified, the figure would be scaled equally in both dimensions. By specifying both a `height` and a `width`, figures can be scaled disproportionately, with interesting results.

There are a few caveats associated with using `psfig`:

- For `psfig` to find the natural size of a figure, the figure must have a proper bounding box comment; see previous bounding box discussion.
- Some versions of L<sup>A</sup>T<sub>E</sub>X will fail to center a lone figure properly in a center environment; a good work-around is to precede the figure with a hard space, *e.g.*

```
\begin{center}
\psfig{figure=...}
\end{center}
```

- On very large documents with many figures, the printer memory allocated to DVIPS may have to be limited; refer to ArborText documentation for setting LaserWriter memory.
- The `\psfig` macro will be confused by extra white space or new lines in its argument, *e.g.*

```
\psfig{figure=piechart.ps, height=1.5in}
```

causes `psfig`'s parsing routine to terminate at the space; L<sup>A</sup>T<sub>E</sub>X will interpret the `height=1.5in` as text. Long `psfig` command lines may be split using % line terminators, *e.g.*

```
\centerline{\psfig{figure=piechart.ps,
height=1.5in,width=2.3in,clip=}}
```

Certain PostScript figures (such as large bitmap images being transmitted at 9600 baud) can tie up a slower PostScript device such as an Apple LaserWriter for quite some time. To circumvent this, a figure may be printed in draft mode, which will reserve the same space on the page, but will print just the name of the file<sup>8</sup> from which the figure is derived and not actually include it. The macro `\psdraft` will switch into draft mode, and all subsequent `psfig` macros will produce draft figures. The macro `\psfull` will switch out of draft mode.

The preceding discussion of `psfig` is appropriate for PostScript graphics generated by Cricket Draw and *any* other type of EPS file. The only exception to this broad statement is a QuickDraw file generated by the Command-F option. QuickDraw files require a suitably modified LaserPrep file to be prepended to the QuickDraw file. `psfig` provides an easy mechanism for including the LaserPrep file as described below.

There is a `psfig` macro `prolog` option for specifying a file that should be prepended to the figure. The name of the prolog is, of course, site dependent; we have used `lprep68.pro`. For example, if you had a file `frog.mac` that contained the QuickDraw to draw Kermit (The Frog), he could be included with:

```
\psfig{figure=frog.mac,prolog=lprep68.pro}
```

If there are many such figures, it is probable that the repeated inclusion of the `prolog` file will cause a significant increase in the size of the print file and its transmission time. An alternative method is to load the `prolog` file once globally, so that it will be available throughout the rest of the document. Use

```
\psglobal{lprep68.pro}
```

at the beginning of your document to achieve this effect. For this to work properly, the `\psglobal` must appear before any Macintosh figures, and the final output must not be page reversed.<sup>9</sup>

Recent experience has shown that the use of `\psglobal` with the ArborText DVIPS driver conflicts with Macintosh graphics included inside a figure.

---

<sup>8</sup> The current implementation of `psfig` does not support underscores (\_) in file names for draft mode. Since `psfig` places the file name on the page using L<sup>A</sup>T<sub>E</sub>X commands, reserved L<sup>A</sup>T<sub>E</sub>X characters cannot be used in the file name.

<sup>9</sup> A page reversed document prints the last page first and first page last. It is possible to use `\psglobal` in a page reversed document; place it just before the last figure in your document. This is living dangerously, and you do so at your own risk.

ure environment; *i.e.* the figure prints by itself on a page separate from the L<sup>A</sup>T<sub>E</sub>X document. Two solutions are: always include a prolog along with each Macintosh graphics (brute force) or include the \psglobal inside the first figure environment (lucky hague).

#### 4 Conclusion

The combination of a Macintosh for producing high quality graphics and L<sup>A</sup>T<sub>E</sub>X for producing high quality typeset documents is becoming very popular as a 'total' document preparation system in many working environments. The development of tools, such as the psfig macros, that make integrating Macintosh graphics in L<sup>A</sup>T<sub>E</sub>X documents easier, will undoubtedly grow in popularity. It is hoped that the information collected in this article helps more users produce better documents.

#### References

- [1] Hosek, D., *Report from the DVI Driver Standards Committee*, TUGboat, Vol. 10, No. 1, p. 56, April 1989.
- [2] Varian, H. and J. Sterkin, *MacDraw Pictures in T<sub>E</sub>X Documents*, TUGboat, Vol. 7, No. 1, pp. 37-40, March 1986.
- [3] Renfrow, J.T., *Methodologies for Preparing and Integrating PostScript Graphics*, TUGboat, Vol. 10, No. 4 — 1989 Conference Proceedings, pp. 607-626, December 1989.
- [4] Woodruff, B., *PostScript and the Macintosh: A History*, MacTech Quarterly, Volume 1, Number 2, Summer 1989, pp. 119-120.
- [5] *Encapsulated PostScript Files Specification Version 2.0*, Adobe Systems Inc., 1585 Charleston Road, P.O. Box 7900, Mountain View, CA 94039-7900, (415)961-4400, 16 January 1989.
- [6] *PostScript Language Reference Manual*, Appendix C: Structuring Conventions, Adobe System Inc., Addison-Wesley Publishing Co., Inc., 1985, p. 268.

◊ Len Schwer  
APTEK, Inc.  
4320 Stevens Creek Blvd.  
Suite 195  
San Jose, CA 95129  
[micro2.schwer@sri.com](mailto:micro2.schwer@sri.com)

---

#### Combining Graphics with T<sub>E</sub>X on PC Systems with Laser Printers, Part II

Lee S. Pickrell

#### Abstract

In this article we will extend our premise that T<sub>E</sub>X affords an excellent mechanism for combining graphics in T<sub>E</sub>X documents. We propose a method for including graphics that brings to bear the full power and versatility of T<sub>E</sub>X for positioning the graphics as well as the text. The technology for implementing this feature will be discussed, including certain limitations. We will also consider possible benefits of file conversion utilities, particularly the potential advantage of converting graphics to the PK/TFM file format of T<sub>E</sub>X fonts. One application of this feature is that the captured graphics can be used with PostScript drivers. This technique can significantly increase the number of graphics sources available to PostScript-based T<sub>E</sub>X by accessing applications that support the LaserJet PCL language. Finally screen capture will be examined as an adjunct to printer capture in the case that printer capture is not practical.

#### 1 Introduction and review

In our first article [1] we made several assertions, in particular, that T<sub>E</sub>X provided a natural platform for mixing graphics with typeset text. Several graphic plots were included that were obtained from different application programs (several more will be included in this article), which we hope substantiated our assertions:

- T<sub>E</sub>X provides a natural platform for graphics insertion, certainly comparable to any other word processing system.
- T<sub>E</sub>X has suffered from a *perception* that it does not handle graphics well, probably grounded more in psychology than technical reality, and possibly due to the broad spectrum of computing systems and distinct device driver programs over which T<sub>E</sub>X is implemented.
- The IBM PC and LaserJet printer are the logical *starting* place for demonstrating the graphics capabilities of T<sub>E</sub>X because graphics applications for the PC/LaserJet combination have become ubiquitous.
- Printer output capture is the best method for obtaining graphics images because the available resolution is much higher than screen capture and the number of graphics sources is much larger than file conversion.

## 2 Graphics positioning, bringing the full power of *TeX* to bear



The thesis of this article and the premise of the **CAPTURE** design is that graphics images should be manipulated by *TeX* with the same facility that typeset text is positioned. This assertion is both practical and consistent with the *TeX* design philosophy. *TeX* is considered a "document preparation system" [2, 3], therefore it should have control over all the contents of the typeset page. Moreover, if graphics are included in a typeset document, the graphics and text should be combined in some harmonious fashion, or the document will be neither aesthetic nor readable.

It is quite possible to regard a graphics image in *TeX* as the functional equivalent of a font of type. As an example, the graphic at the beginning of the preceding paragraph was captured using **CAPTURE** and inserted using the code segment:

```
\drop{\insertplot{logopcl}{1}{1.12}}
```

The `\drop` macro is defined in the `drop.sty` file which is available in the public domain [4]. It was designed to start a paragraph with a dropped, large letter of type, as is done with early Bibles and such. The `\drop` macro manipulated the graphic image just as it would any other font of type. The purpose of this illustration is to demonstrate that a properly processed graphics image can be treated identically to a font of type; indeed, text and graphics can be indistinguishable for *TeX* operation.

An ancillary benefit of this approach is that the artificial distinction between *TeX* and the device driver programs is reduced. Perhaps the perception that *TeX* does not handle graphics well stems from the somewhat artificial separation of *TeX* from the device driver programs [1]. Although graphics must be included on the device driver level, the distinction is less severe if *TeX* controls the location and space for the graphics.

### 2.1 File processing and macro definitions

*TeX* will be able to manipulate a graphics element if it is operationally equivalent to a "box". A box in *TeX* is a typographic unit, which on the most fundamental level is an indivisible character of type [3]. Two requirements must be satisfied to establish a graphics image as the equivalent of a *TeX* box:

- The space required for the graphics must be defined as an `\hbox` (`\mbox` in *LATEX*), with the same dimensions (height and width) as the actual graphics.

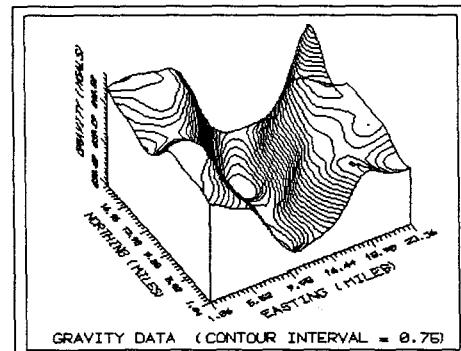
- The graphics must be positioned inside of the box.

A box with the proper horizontal and vertical dimensions can be created with a simple *TeX* macro. Once defined, it can be manipulated like any other box in *TeX*. A typical definition for the `\insertplot` command used in **CAPTURE** [5] is:

```
\def\insertplot#1#2#3{%
  \vbox to #2 true in {
    \vfill
    \hbox to #3 true in
      {\special{pcl:#1} \hfill}
  }% End of vbox
} %End of Definition
```

The `\insertplot` command is functionally equivalent to an `\hbox` in *TeX* (an equivalent form makes an `\mbox` in *LATEX*). It creates a box with the exact height and width of the graphics, specified by the 2<sup>nd</sup> and 3<sup>rd</sup> parameters, which are obtained from **FIXPIC** after processing the captured graphics file. For example, the plot in Figure 1 was created using the following code segment:

```
\begin{figure}[htb]
\begin{center}
\fbox{\kern 1pt \fbox{%
\insertplot{surfpcl}{1.64}{2.13}}}
\caption{This is output ...}
\label{surfpcl}
\end{center}
\end{figure}
```



**Figure 1:** This is output from the demonstration diskette of the **SURFER** scientific data plotter by Golden Software, Inc. A frame has been drawn around the plot to show the "box" which *TeX* manipulates. When the graphics file has been properly processed, the box defines the location of the graphics, and the image is surrounded by the frame.

This macro is essentially the definition for the `\pfig` command contained in the `plot.sty` file

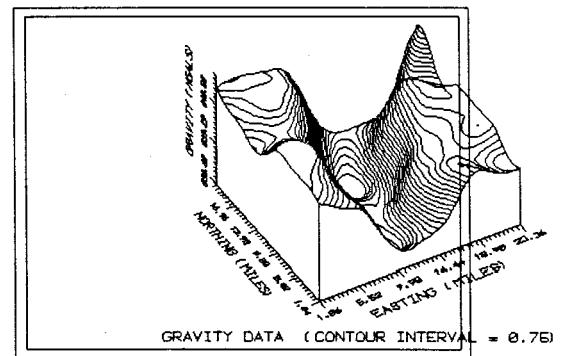
which is part of **CAPTURE**. It differs by the use of the `\fbox` command. `\fbox` is a **LATEX** macro which creates a box around the the text parameter and surrounds it with a frame [2]. In this instance, the `\insertplot` box is substituted for the parameter and a frame is drawn around it. The use of the frame command again emphasizes that the `\insertplot` command is functionally identical to an `\hbox` in **TEX**. The `\fbox` command drew a frame around the graphic just as it would around any text block. This construct also conveniently highlights the position where **TEX** thinks the graphics plot is located.

It is immediately apparent viewing Figure 1 and the macro defined above, that the graphics image is relocatable and has been positioned by **TEX**. The insertion macro uses the `\begin{center}` ... `\end{center}` environment to center the plot in the current **TEX** context. Because *TUGboat* is typeset in a two column format, the image is centered in a column, as it should be. However, if this same article were typeset in a single column format, the plot would be automatically centered in a page.

The `\insertplot` command also contains the `\special` command which instructs the device driver to load and print the graphics file at the present cursor location. In order for the graphic to be positioned properly inside the `\hbox` (inside the frame), the location of the graphic must be well defined with respect to the `\special` command and the `\special` command must have a well defined location inside the box. Fortunately, the definition of the `\special` command from Knuth [6] specifies that it will have a unique, well defined location on the page: "Therefore it is implicitly associated with a particular position on the page, namely the reference point that would have been present if a box of height, depth, and width zero had appeared in place of the `whatsit`" [6]. If the position of the graphic is linked to the location of the `\special` command, it will also have a well defined location that can be placed inside the `\hbox`.

The technology for connecting the graphics to the `\special` command is based on the LaserJet command structure. The LaserJet PCL language contains a control code which says in effect: "start the graphic at the present cursor location" [7]. If the graphics file contains this control code, then the image will be inserted starting at the location of the `\special` command, which is well defined, and will be centered inside the `\hbox` defined by the `\insertplot` command. This is the technique used by **CAPTURE** to allow **TEX** to manipulate graphics in the same way as text.

An important requirement however, is that the graphics file contain only this relative positioning command (start the graphic at the present cursor location) *and no other positioning commands*. The reason is that other positioning commands will either override the relative position command, or change the cursor location so that it no longer is coincident with the location of the `\special` command. As an example, the graphic in Figure 2 was inserted using the identical commands as Figure 1. However, the file in Figure 2 was not processed to remove the additional positioning commands. The result is that **TEX** still thinks there is an `\hbox` containing the graphic, and even draws a frame around it. However, the graphics image is offset relative to the `\special` command so the image does not appear inside the frame. For this reason **CAPTURE** contains the **FIXPIC** utility which is run automatically after every graphics capture. **FIXPIC** removes all of the positioning control sequences except the relative position command, which it may insert if necessary.



**Figure 2:** This figure is the same as Figure 3, *except* that the positioning commands have not been removed from the image file. The frame has again been drawn where **TEX** *thinks* the graphic is located. However, because the graphics file was not properly processed, additional positioning commands remain, and the image is offset.

The example shown in Figure 2 is relatively benign. The positioning sequences in the graphics file specified a relative position, so the image is offset somewhat from the position of the `\special` command. However, it is more common to find *absolute* positioning commands in graphics files. These commands simply place the graphic image at some fixed location on the page, and ignore entirely the present cursor position [7]. Therefore there would be no correlation at all between the actual location

of the graphics and where *T<sub>E</sub>X* thinks the graphics are located.

All of the application programs we have tested, which provide LaserJet graphics, have used absolute positioning commands. This choice is logical. An application program has no way of knowing *a priori* the location of the LaserJet cursor. Attempting to write the graphics at the present cursor location would be dangerous, because the graphics could appear anywhere on the page. Conversely, the developers of these programs probably want the graphics to be somewhat centered, and they can control the graphics position unambiguously with absolute coordinates.

The problem of converting these captured graphics files into a *T<sub>E</sub>X*-compatible format is complicated because there are several different LaserJet positioning codes. There are 2 codes for specifying either graphics start at the cursor or at the left hand side of the page, 6 relative positioning codes (relative to the old cursor position) and 6 code sequences which place the cursor at an absolute location on the printed page. All must be removed from the graphics file (except the relative positioning command code) without disturbing any of the graphics data.

Another problem is the use of additional vertical white space by some graphics applications. White space is simply a series of null data transfers before or after the graphical image. Most of the application programs we tested added some extra white space around the image. These programs make no assumptions that the images produced might eventually be included in *T<sub>E</sub>X* documents, so the additional white space may have been included for convenience. In some extreme cases, white space was used to position the graphics on the page as an alternative to the position commands. If it is not removed, the additional space will distort the page layout and aesthetic appeal of the document, pushing the rest of the text and graphics far from a particular plot. In extreme cases it can force a premature page eject. Another function of **FIXPIC** is to remove all leading and trailing white space in a graphics file. Spacing between the graphics and the text can then be determined to satisfy aesthetic appeal, and is controlled by *T<sub>E</sub>X*.

## 2.2 Absolute coordinates: the exception to *T<sub>E</sub>X* positioning

Unfortunately, some graphics files cannot be processed so as to be relocatable. There are applications which use absolute positioning commands throughout the graphics file instead of just at the

beginning. The reason is that absolute positioning commands can considerably reduce the size of the file and the concomitant time to print. This issue is important for LaserJet printers without additional memory space. Most applications which we have tested do not use this method; rather, white space is used to position the LaserJet cursor. This method requires a larger file and more time to print, but the entire image can be moved by changing the position of the cursor at the start of printing.

When absolute position commands are embedded throughout the file (as opposed to being placed at the beginning only), the **CAPTUREd** file cannot be positioned by *T<sub>E</sub>X*. The graphics can be captured and included in a *T<sub>E</sub>X* document, but the position will be determined by the application program. If an attempt is made to force relocation, the image will be distorted because parts of the image will be placed at different locations. **CAPTURE** has an option which enables absolutely positioned graphics to be included in *T<sub>E</sub>X* without distortion; however, *T<sub>E</sub>X* cannot control the position of the graphics. The application program must specify the plot location to be the proper position for the *T<sub>E</sub>X* document.

This problem is not entirely intractable. Most programs which use absolute position coordinates do so for only the horizontal coordinate. The vertical position is specified only once at the beginning of the file, and will be removed by **FIXPIC**. If the desired horizontal position of the graphics can be easily defined (centered for example), then the application program can generally achieve the proper position. The plot will appear at the same location in *T<sub>E</sub>X*, and will have a vertical position depending on its location in the file.

We hope to address this problem further in a future release, by translating an absolutely positioned file to a relatively positioned one.

## 3 The PK/TFM format



The logic that graphics should be functionally equivalent to text can be extended by converting a graphic image file from the LaserJet PCL language to the PK and TFM formats which are specific to *T<sub>E</sub>X* [8, 9]. A graphic image in the combined PK/TFM format isn't *equivalent* to a font of type, by definition it *is* a font of type. For example, the graphic at the beginning of this paragraph was generated by converting the graphic seen earlier from the PCL format to the PK/TFM format using the **CONVERT** utility in **CAPTURE**. It was inserted using the same \drop macro described earlier and the code sequence:

```
\font\largefont=logo
\drop{a} The logic that ...
```

The immediate benefit of this approach may not be clear. Operationally, there seems to be little difference between using the `\insertplot` command and the PK/TFM format, except that the PK/TFM format may be less convenient: two files are created instead of one. However, the advantages of using a PK/TFM version of a graphics file accrue from its device independent nature. For example, none of the commercial *TeX* page previewers will display graphic images included with the `\special` command. However, graphics in the PK/TFM format can be viewed, although some previewers we have tested have memory limitations for large images.

Another benefit is that **CAPTURE** can be used to supplement the graphics for other systems (non-LaserJet). We have argued that a graphics capture utility is unnecessary for PostScript-based systems, because the PostScript language describes both text and graphics and the two can be mixed easily. However, the graphics sources are limited because there are relatively few applications on PCs which support PostScript, due to the high cost, and these tend to be concentrated in the desk-top publishing area. Fortunately, many of the PostScript drivers for PC based systems use the same PK/TFM font files as the LaserJet drivers [10]. Therefore, captured graphics files which are converted to the PK/TFM format can be used with PostScript drivers for *TeX*. The domain of graphics sources for inclusion in PostScript is increased considerably because far more applications support the LaserJet PCL language than support PostScript.

This idea can be generalized by realizing that the PK/TFM format provides a level of device independence, one of the hallmarks of the *TeX* design [3]. Once a PK/TFM file pair has been created, the graphics should be usable on any system that uses the same resolution (300 dots per inch). This set includes the LaserJet systems for which **CAPTURE** was originally targeted, screen previewers, and PostScript systems. We have yet to fully test this idea on other 300 dpi drivers, say for the HP DeskJet, but the idea is intriguing and has been tested on PostScript drivers [10].

This approach also suggests a general design path for future extensions. We have argued that separate **CAPTURE**-like programs may be necessary for each computer/printer combination. However, the PK/TFM standard provides a level of device independence such that a **CAPTURE**-like program may be needed only for each computer/*resolution* combination.

### 3.1 Natural conversions for a *TeX* graphics system

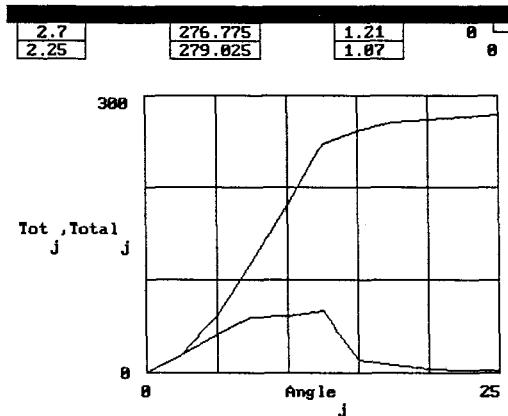
The benefits of a PK/TFM conversion utility also suggest other areas where file format conversion may be useful. File conversion generally does not offer a new source of graphics because it is somewhat redundant with printer or screen capture. However, converting from the LaserJet PCL language to the formats of graphical drawing or paint programs would be useful. The application program would still generate the graphic image, saving the user a considerable amount of work, but the captured graphics could be edited into a final form before inclusion in a document. As a test of this idea, the **CAPTURE CONVERT** utility will convert to the PC Paintbrush PCX format [5].

#### 4 The case for limited screen capture

We maintain that the best method for obtaining graphics is printer capture because of the large source of graphics at high resolution. File format conversion generally offers few *additional* graphics sources, and screen capture generally provides low resolution. However, there are two cases which we have identified in which printer capture is not practical. Both are apparently quite unusual. The first case is an application program which mixes text and graphics in the printer output. An example is the scientific program MathCAD, by MathSoft, Inc.

Printer capture is not practical with MathCAD because it mixes text characters with the graphics output. Although the output can be captured, the resulting file cannot be inserted into a *TeX* document and retain the original likeness. The solution in this instance is to use the screen capture utility, **CPTS**, included with **CAPTURE**. **CPTS** captures the screen image and writes it to a file that can be included in a *TeX* document [5]. Screen capture effectively converts the text characters into a graphic representation because they are displayed in graphics mode and can be resolved into individual pixels. *Moreover, there is no loss of resolution.* The MathCAD printer output is a direct image of the screen display. An example of screen capture from MathCAD is shown in Figure 3.

Other examples are programs which use installable device drivers in the DOS `config.sys` file. Fortunately, this construct is rare, because device drivers for a specific application remain in memory, attached to the operating system regardless of whether the particular application is being used. Valuable memory is wasted and the operating system is cluttered. However, some applications use



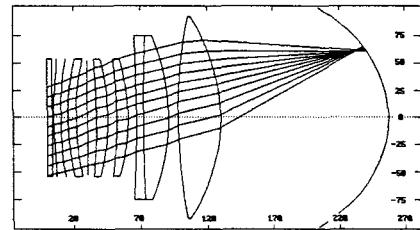
**Figure 3:** This plot is a screen capture from the program MathCAD, by MathSoft, Inc. The original screen was CAPTUREd, converted to the PCX format by CONVERT, edited on PC Paintbrush, and converted back to the PCL format. It was then modified by FIXPIC to have a resolution of 150 dots per inch, producing a plot with reasonable resolution which fits nicely inside the columns of *TUGboat*.

this approach because it provides a uniform interface to all display monitors and hard copy devices. An example is the lens design program, OPTEC-II/87 by SCIOPT Enterprises. The CAPTURE printer utility is unable to capture the printer output because OPTEC bypasses both DOS and the BIOS for the printer output. However, the screen capture utility works fine and there is no loss of resolution. Because the OPTEC interface uses a common set of device drivers, a single raster image is maintained in the program. The output to the printer is derived from the same raster image as the output to the screen; the only difference is which device driver is invoked. Therefore, the screen capture acquires the same image as the printer output. An example of an OPTEC-II/87 image is shown in Figure 4.

The intent of this discussion is to acknowledge that screen capture is a necessary utility for a general graphics capture system. Although it is not the best method in most cases, there are instances when it is the only method that will work.

#### 4.1 Memory management and the terminate and stay resident option

The last issue to be considered is the general architecture of a graphics capture utility in the MS-DOS environment. This issue is not applicable to the general T<sub>E</sub>X graphics problem, but is entirely specific to the IBM/DOS implementation. A distinct limitation of DOS is the 640k memory limit, which



**Figure 4:** This is a plot of a high numeric aperture wide field of view lens. The plot was obtained from the lens design program: OPTEC-II/87 by SCIOPT Enterprises.

has been a particular nemesis for large, complicated, programs. A premise of the CAPTURE design was that it could be used with large application programs, and that both CAPTURE and an application program would occupy memory simultaneously. Therefore, memory size became an important issue.

These considerations lead to a design for CAPTURE which minimizes the use of memory *while the application program is running*. For example, the postprocessing phase is explicitly removed from the image capture routine, the printer and screen capture routines are kept to the minimum size possible, and as many features as possible are incorporated into the postprocessing program. Also, the postprocessor is spawned by the image capture routines only *after* the application program has exited. Finally, CAPTURE does not use a terminate-and-stay-resident (TSR) design. Although a TSR will take no more memory than a normal program, a TSR must be explicitly removed from memory before the storage is released. If any programs are loaded after the TSR, memory can be fractured and the released storage is not contiguous with remaining memory.

Instead, the sequence of operation for CAPTURE runs as follows. The image capture routines modify the operating system in order to capture graphics output and then *spawn* the application program. The application program runs and presumably attempts to output graphics to the printer. Once the application program exits, CAPTURE loads the postprocessor program, FIXPIC. When all processing is complete, CAPTURE meticulously returns the operating system to its original state, releases all its memory, and exits. The memory used is a minimum and is never fractured.

#### 5 Conclusions

We have tried to extend our initial thesis that T<sub>E</sub>X provides an excellent medium for including graphics

with text. In the case of the PC/DOS implementation in particular, graphics applications are ubiquitous so there is a wide array of graphics sources. Moreover, we have suggested a *method* for including graphics with TeX that allows TeX the same control over graphics images as fonts of type. This approach affords a seamless blend of graphics and text in the same document. The distinction between device driver and TeX is softened. Although the graphics insertion occurs at the device driver level, the *control* is retained in TeX.

This idea has been extended to include the notion of converting graphics files to the PK/TFM format of TeX. The primary benefit of this approach is expanding graphics capture to TeX implementations which do not use the LaserJet printer. In particular, CAPTURE can support PostScript drivers for TeX that use the same computer modern fonts in the PK/TFM format as the LaserJet drivers. The range of graphics sources available to PostScript users is considerably increased over the range of applications which presently support PostScript. Other extensions may also be possible.

We have consistently emphasized that CAPTURE serves as an example and proof-of-principle that the graphics capability of TeX is considerable. We would like to propose (hopefully without being presumptuous) that other graphics implementation programs adopt some of the ideas discussed here. For example:

- TeX should be able to manipulate graphics images equivalently to fonts of type.
- A graphics program for TeX should support the PK/TFM format to maintain the greatest possible device independence.

In this way, the distinction between graphics and text in TeX should be diminished and a connection between the various implementations of TeX can be maintained by the device independent nature of the standard TeX formats.

## References

- [1] Lee S. Pickrell. "Combining Graphics with TeX on IBM PC-Compatible Systems with LaserJet Printers." *TUGboat*, 11(1):26 – 31, 1990.
  - [2] Leslie Lamport. *LATeX, A Document Preparation System, Users Guide & Reference Manual*. Addison-Wesley Publishing Company, Reading, Mass., 1986. ISBN 0-201-15790-X.
  - [3] Donald E. Knuth. *The TeXbook*. Addison-Wesley Publishing Company, 1986. ISBN 0-2-1-13448-9.
  - [4] David G. Cantor. "DROP.STY." Published in TeXhax, number 16, 1988. Available on the Clarkson Archive Server (public domain).
  - [5] CAPTURE, *A Program for Including Graphics in TeX*. Wynne-Manley Software, Inc., 1094 Big Rock Loop, Los Alamos, NM 87544, March 1990.
  - [6] Donald E. Knuth. *The TeXbook*, pages 228–229. Addison-Wesley Publishing Company, 1986. ISBN 0-2-1-13448-9.
  - [7] *LaserJet series II User's Manual*. Hewlett Packard Corporation, Boise Division, P.O. Box 15, Boise, Idaho 83707, December 1986. Part No. 33440-90901.
  - [8] David Fuchs. "TeX Font Metric Files." *TUGboat*, 2(1):12 – 17, 1981.
  - [9] Tomas Rokicki. "Packed PK Font File Format." *TUGboat*, 6(3):115 – 120, 1985.
  - [10] *TEXPRINT/PS User Guide*. Oregon House Software, Inc., 12894 Rices Crossing Road, Oregon House, CA 95962, 1988.
- ◊ Lee S. Pickrell  
Wynne-Manley Software, Inc.  
c/o Micro Programs, Inc.  
251 Jackson Ave.  
Syosset, NY 11791  
(516) 921-1351

## Resources

### Data General Site Report

Bart Childs

The distribution with the new versions of TeX and METAFONT is nearly finished. We are also rewriting the drivers for the DG, QMS, and LaserJet printers in Silvio Levy's CWEB. We have decided to use Tom Rokicki's PostScript drivers.

◊ Bart Childs  
Dept. of Computer Science  
Texas A&M University  
College Station, TX 77843-3112  
[bart@cssun.tamu.edu](mailto:bart@cssun.tamu.edu)

---

## VM/CMS Site Report

Joachim Lammarsch

My first report as new VM/CMS site coordinator starts with bad news. I have heard that it is possible to install a virus into IBM DCF or Waterloo Script input using the command .sy. This is the vehicle to send commands to CMS. Within the regular VM/CMS version of *TeX* it's possible to use the command \cms to do the same. Therefore the warning: Be careful *texing* strange input; first look for the command \cms! I haven't heard anything about viruses in *TeX* input yet, but nevertheless I'll try to find a method to make this kind of virus impossible.

Now the good news: Peter Breitenlohner has finished his work and sent the new *TeX* 3.0 to me. It contains not only *TeX* 3.0, but also METAFONT 2.0, VPTOVF, VFTOVP and last, but not least, a Big*TeX* 3.0 containing two times more memory words than the normal version. Many thanks, Peter!!

Ferdinand Hommes from GMD, Gesellschaft für Mathematik und Datenverarbeitung Bonn, has sent me new public domain drivers for IBM laserprinters supported by PSF and for IBM 4250; for PostScript printers; QMS Lasergrafix model 800, 1200, and 2400; and for IBM display stations supported by GDDM. Unfortunately, there are only text files.

Dean Guenther has sent me DVIALW, a driver ported to VM/CMS by S. Sathaye from Nelson Beebe's public domain dvialw driver.

I plan to bring the new distribution tape with me when I come to TUG90. It should be available from Maria Code in July.

I have created a new discussion list named *TeX-IBM* to discuss problems concerning the implementation of *TeX* and his children under VM/CMS. All IBM MVS users are invited to join this list, too. To subscribe to the list, send the command

**SUB *TeX-IBM* *firstname* *familyname***  
to your nearest listserv.

◊ Joachim Lammarsch  
Computing Center  
University of Heidelberg  
Im Neuenheimer Feld 293  
6900 Heidelberg 1  
West-Germany  
Bitnet: X92@DHDURZ1

---

## Resources Available to *TeX* Users

Barbara Beeton and Ron Whitney

In this installment we have a few updates to the inaugural column in *TUGboat* 11, no. 1.

### Archives with network connections

We have received several lists of network hosts with a summary of items that can be found at each. Unfortunately, time has prevented our checking the data (it is clearly out of date, as Score.Stanford.edu appears in every list). So, rather than spread erroneous information, we will spend some time over the summer checking it and provide an accurate list in the fall. Anyone who would like to assist with this research, or knows of any interesting repositories, please get in touch.

### Sources of software and macros for PC and Macintosh

The following information was posted recently to UK*TeX* (issue 14) by Sebastian Rahtz:

The Aston *TeX* archive has a new version of *TeX* for MS-DOS and OS/2, contributed by Eberhard Mattes from Stuttgart. This release comprises all of *TeX*, METAFONT, support programs (BIB*TeX*, Makeindex, webware, etc.), dvi drivers, previewer, and drawing package for L<sup>A</sup>T<sub>E</sub>X pictures (*texcad*).

On the good side:

- there are separate binaries for normal MS-DOS, for 286/386 processors (which make things go a little faster), and for OS/2 protected mode
- the release has a full METAFONT
- expanded memory is used if present
- there are 'big' versions of *TeX* and METAFONT; the former is a boon for L<sup>A</sup>T<sub>E</sub>X users who load lots of extra macros (such as P<sub>I</sub>C<sub>TE</sub>X)
- the *TeX* is as fast as, if not faster than, sb*TeX*
- the printer driver family and screen preview share a common interface, and a common set of \special commands for graphics (used in the *texcad* package); the previewer can use the same 300 dpi fonts as a laserprinter. An interesting development is the use of optional 'libraries' of fonts, a convenient way of combining together those huge directory hierarchies.

On the down side:

- the documentation is all in German. And why not, you may ask? It only matters to the

uneducated among us! [Editor's note: but see below.]

- the printer drivers are for dot matrix and laser printers, but *not* for PostScript
- *T<sub>E</sub>X* 3.0 isn't available yet (but promised soon)
- the huge *T<sub>E</sub>X* is rather slow

If any or all of these apply to you:

- you have a reading knowledge of German
- you want *T<sub>E</sub>X* for OS/2
- you need a big *T<sub>E</sub>X*
- you need a fast free PC METAFONT

then you should check out em*T<sub>E</sub>X*. Users who just want a good, fast *T<sub>E</sub>X* may be better off getting Wayne Sullivan's excellent sb*T<sub>E</sub>X*, now in *T<sub>E</sub>X* version 3.0, as it does nearly all you want and is a little simpler to set up. But em*T<sub>E</sub>X* is an excellent way to set up a complete *T<sub>E</sub>X* on your PC.

The files are a set of B00-encoded .zip archives in

[tex-archive.tex.msdos.emtex]

at Aston University.

**Editor's note:** A later issue of UK*T<sub>E</sub>X* has announced that most of the documentation has now been translated into English, and is also available from the Aston archive. Instructions for obtaining information and files from Aston have appeared most recently in *TUGboat* 10, no. 2, pages 194–195, and can also be found at the end of every issue of UK*T<sub>E</sub>X*.

We have also learned that em*T<sub>E</sub>X* is available for anonymous ftp from

terminator.umich.cc.edu

in the directory soft/text-mgmt/emtex.

### Electronic discourse

**TEX-D-L.** The last issue contained two errors with respect to this list—in the name and in the node. The correct form of the name is shown here, with two hyphens and no underscores. To subscribe, send a message to LISTSERV@DEARN.Bitnet containing the request

SUBS TEX-D-L *(your name)*

This list is conducted in German.

**TEX-D-PC.** A second list in German has been established for those interested in matters concerning *T<sub>E</sub>X* on PCs. Send the command

SUBS TEX-D-PC *(your name)*

to LISTSERV@DHDURZ1.Bitnet.

---

### A Proto-TUG Bibliography: Installment Two

Barbara Beeton

In the last issue, we presented the first installment of a TUG bibliography, in progress, containing references to books and articles about *T<sub>E</sub>X*, *L<sub>A</sub>T<sub>E</sub>X*, *WEB* and related topics, or prepared using one of these tools. The list that follows continues with references to additional works that had accumulated in my office as well as citations sent in by obliging *TUGboat* readers. Please send more.

We have not yet created a bibliography style especially for *TUGboat*, so some of the elements we would like to show are hidden. This will be corrected as soon as we find time. (And have received the newest, "final", version of *BIB<sub>T</sub>E<sub>X</sub>*, now under construction.) In the meantime, if you send in information, please include the following:

- Author(s), full name(s)
- Title
- For books, including proceedings or other collections:
  - Publisher, with address
  - Year of publication
  - ISBN
  - Editor (for collections)
  - Series name and number, if relevant
  - Conference name, location and date (for proceedings)
- For journals prepared completely or substantially in *T<sub>E</sub>X*:
  - Year when publication in *T<sub>E</sub>X* began; year when journal began publication, if not the same
  - Publisher and address
- For articles in journals:
  - Year and month of publication
  - Volume, issue and page span
- For articles in collections:
  - Full reference for the collection as a whole: editor, title, publisher, conference information, etc.
  - Page span of article
- For technical reports:
  - Publisher or sponsoring institution, with address
  - Series name and number
  - Year and month of publication

- Indication of extent to which  $\text{\TeX}$  (or  $\text{\LaTeX}$ ,  $\text{\AMS-\TeX}$ , etc.) was used in preparation
- Any other useful information, e.g. translation, language

Although the present compilation is in  $\text{BIB}\text{\TeX}$  format, I have been reminded of the existence of another competent bibliographic tool,  $\text{Tb}$ . (See the article by James Alexander in *TUGboat* 8, no. 2.) The suggestion has been made that the bibliography be maintained in parallel in both forms, and we are seriously considering doing just that, when time permits.

Once again, please send your suggestions and candidates for inclusion.

Here is the second installment, in two parts: publications about  $\text{\TeX}$ , and publications prepared with  $\text{\TeX}$ . In both sections, preparation with  $\text{\TeX}$  is assumed unless stated otherwise.

### Publications about $\text{\TeX}$

- Paul W. Abrahams, with Karl Berry, and Kathryn A. Hargreaves.  *$\text{\TeX}$  for the Impatient*. Addison-Wesley, Reading, MA, 1990.
- Wolfgang Appelt.  *$\text{\TeX}$  für Fortgeschrittene*. Addison-Wesley Verlag, Bonn, 1988.
- Malcolm Clark (editor).  *$\text{\TeX}$ : Applications, Uses, Methods. Proceedings, Third European  $\text{\TeX}$  Conference,  $\text{\TeX}88$ , Exeter, August 1988*. Ellis Horwood, Chichester, 1990.
- Jost Krieger and Norbert Schwarz. *Introduction to  $\text{\TeX}$* . Addison-Wesley Europe, Amsterdam, 1990. Translation of *Einführung in  $\text{\TeX}$* .
- Norbert Schwarz. *Einführung in  $\text{\TeX}$* . Addison-Wesley Verlag, Bonn, 1988(?)
- Norbert Schwarz. *Inleiding  $\text{\TeX}$* . Addison-Wesley Europe, Amsterdam, 1990. Translation of *Einführung in  $\text{\TeX}$* .

### Publications prepared with $\text{\TeX}$

- Harold Abelson and Andrea A. diSessa. *Turtle Geometry*. MIT Press, Cambridge, MA, 1981. This book was prepared with  $\text{\TeX}80$ ; this was described in *TUGboat* 2, no. 3 in an article by Michael Sannella.
- Ronald N. Bracewell. *The Hartley Transform*. Oxford University Press, 1986.
- Commentationes Mathematicae Universitatis Carolinae. This publication was prepared with  $\text{\AMS-\TeX}$ .
- Gerard Gazdar, Alex Franz, Karen Osborne, and Roger Evans. *Natural Language Processing in the 1980s, A Bibliography*, volume No. 12 of *Center*

*for the Study of Language and Information Notes*. U. Chicago Press, 5801 Ellis Ave., Chicago, IL 60637, 1987. This book was prepared automatically from a database and set with  $\text{\TeX}$ . The intent is to keep it updated.

- R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics*. Addison-Wesley, Reading, MA, 1989.
- Alan Hoenig. *Applied Finite Mathematics*. McGraw-Hill Publishing Company, New York, 1990.
- Arthur M. Keller. *A First Course in Computer Programming Using PASCAL*. McGraw-Hill Publishing Company, New York, 1982.
- Steven E. Koonin and Dawn C. Meredith. *Computational Physics*. Addison-Wesley, Redwood City, 1990.
- Tom Lyche and Larry L. Schumaker. *Mathematical Methods in Computer Aided Geometric Design*. Academic Press, Boston, 1989.
- Roman Maeder. *Programming in Mathematica*. Addison-Wesley, Reading, MA, 1990.
- William H. Press et al. *Numerical Recipes*. Cambridge Univ. Press, Cambridge, 1986. Originally for FORTRAN and then rewritten for a C version, *Numerical Recipes in C*; also example books in Fortran, Pascal and C, making a total of five books with “Typeset in  $\text{\TeX}$ ” on the back of the title page.
- Robert Sedgewick. *Algorithms*. Addison-Wesley, Reading, 1988.
- Robert Sedgewick. *Algorithms in C*. Addison-Wesley, Reading, 1990.
- J. F. Traub, G. Wasilkowski, and H. Woźniakowski. *Information-Based Complexity*. Academic Press, New York, 1988. This book was prepared with  $\text{\AMS-\TeX}$ .
- Stephen A. Ward and Robert H. Halstead, Jr. *Computation Structures*. The MIT electrical engineering and computer science series. MIT Press, Cambridge, MA, 1990.
- Stephen Wolfram. *Mathematica: A System for Doing Mathematics by Computer*. Addison-Wesley, Redwood City, 1988. This book was prepared with  $\text{\TeX}$ ,  $\text{\LaTeX}$  and PostScript.
- Daniel Zwillinger. *Handbook of Differential Equations*. Academic Press, Boston, 1989.

---

## New Books on *TEX*

Victor Eijkhout

There is a piece of good news to be reported: two new books on *TEX* have appeared recently, one for beginning to intermediate users, and one for intermediate to advanced users. And there's more good news: *TEX* is so widely spread that both books originated in Germany, and are written in German. Of the introductory book, translations into English and Dutch exist, but the advanced book, in more than one respect the more interesting of the two, has not been translated yet.

*TEX* books in languages other than English are a good thing for two reasons. One is that they give an indication of the widespread use of *TEX*. The other is that, to quote Norbert Schwarz, author of *Einführung in *TEX** [1], such books are 'a bit more internationally oriented than a book of English or American origin would probably be'. This is especially apparent in *TEX für Fortgeschrittene* [2] by Wolfgang Appelt, which has a whole chapter on 'Deutschsprachige Text', containing useful remarks that are relevant to more languages than just German.

### Introduction to *TEX*

*Introduction to *TEX** by Norbert Schwarz assumes no knowledge of *TEX* whatsoever; indeed the first chapter 'General information' gives a short list of the merits of *TEX*. This makes for a nice and motivating introduction for the complete novice.

The same holds for chapter 2, 'Operation', that contains, after a few pages of braces, backslashes, and punctuation, a first example of the use of *TEX*. Some thirty commands are used here. Obviously the author wants to get the reader going: the details will come later.

Chapter 3 was written in the same vein. In 30 pages a large amount of information about 'Setting text' is given to the reader, with lots of examples. However, this chapter had me frowning a number of times. It is the author's style of writing to use unusual examples like

```
{\obeylines\everypar{\hfil}...}
```

to introduce the concepts of `\everypar` and `\parfillskip`, but it wouldn't be mine. And I object to

```
\centerline{\it The current page has  
the number \folio}
```

(because the statement may be untrue due to asynchronous output routine behaviour).

Fortunately, some important concepts are explained more fully in chapters on macros and 'How *TEX* works' — although I feel that the section on modes is a bit skimpy. There are two nice chapters on mathematical typesetting, there is a short chapter on output routines, and I was particularly pleased with the chapter on 'Tables and alignment' as it is well-written and contains good examples.

The main part of the appendix to this book is an 80 page (!) list of all *TEX* and plain *TEX* commands. The explanations are short, but certainly not cryptic, and often an illuminating example is given. Definitely a good idea of the author.

### *TEX* for the advanced

*TEX for the Advanced* by Wolfgang Appelt is a very different book. The subtitle, 'Programming techniques and macro packages' is probably the best indication to its contents. Wolfgang Appelt argues in the preface the need for high level macro packages, and then sets out to assist the reader in constructing such packages. He does this in three ways.

The preface, the introductory chapter, and a chapter 'Macro packages' give general thoughts on how macro packages should be structured, and what their nature should be. He distinguishes between the logical structure and the layout structure, and, for both of these, the generic and the specific structure. It is useful to have such concepts explained in some detail, and the reader won't hear me arguing the author's point of view.

Pure *TEX* theory is treated in chapters on 'Spaces' — such a chapter must be answering many prayers of desperate *TEX*ers — and 'Macros and parameters'. The author has a very clear style of exposition, but his explanation of conditionals, sufficient for most cases, distorts the truth a bit.

Lastly, four chapters can be classified as "case studies in macro package design." They treat the subjects of a font selection scheme, text structures (lists and sectioning), referencing (including table of contents), and adaptations necessary for the German language. These chapters give complete sets of macros, and they are well explained.

Appelt makes no attempt at being complete. Mathematical typesetting and alignments are not treated in this book, and output routines are hardly touched upon. Given the size of the book this would not have been possible, and concentrating on a few selected topics is probably a good idea.

In all, this book is maybe not sufficient reason to start learning German — which means you'll never make such delightful discoveries as that *ragged right* is *Flattersatz* (*flutter setting*) in German —

but if you know a smattering of the language it certainly won't harm you to pick up this book.

### How does it look?

When a book about *T<sub>E</sub>X* appears, there is an obvious question: "Has it been done in *T<sub>E</sub>X*?" For both books reviewed here the answer is yes, but the results are widely different. The Appelt book is set in 12 point Computer Modern with non-obtrusive headings, which gives a surprisingly open and readable page. Of the Schwarz book I have only seen the Dutch and English translations, which are totally unlike each other. The English translation is set in Computer Modern, photographically reduced to 10 point. Unfortunately, the book was printed rather lightly, which makes the page appear somewhat hazy.

The Dutch branch of Addison-Wesley must have been in an adventurous mood, combining New Century Schoolbook as a text face with Avant Garde headings. Choosing Courier as the typewriter font was not the optimal choice, but the overall result is rather pleasant — even though there have been a few accidents in typesetting the examples.

As a conclusion I would state that both books are an asset to the *T<sub>E</sub>X* community. Neither book is a definite *T<sub>E</sub>X* bible, but niches certainly exist for both to fill.

[1] *Einführung in T<sub>E</sub>X*, Norbert Schwarz, Addison-Wesley Verlag, Bonn 1988(?) ISBN 3-925118-97-7.

*Inleiding T<sub>E</sub>X*, Norbert Schwarz, Addison-Wesley Europe, Amsterdam 1990, ISBN 90-6789-151-7.

*Introduction to T<sub>E</sub>X*, Jost Krieger and Norbert Schwarz, Addison-Wesley Europe, Amsterdam 1989, ISBN 0-201-51141-X.

[2] *T<sub>E</sub>X für Fortgeschrittene*, Wolfgang Appelt, Addison-Wesley Verlag, Bonn, 1988, ISBN 3-89319-115-1.

◊ Victor Eijkhout  
 Department of Mathematics  
 University of Nijmegen  
 Toernooiveld 5  
 6525 ED Nijmegen, The  
 Netherlands  
 Bitnet: u641001@HNYKUN11

---

### DECUS *T<sub>E</sub>X* Collection — Submissions Wanted

Ted Nieland

DECUS is putting out a Call For Submissions to the DECUS *T<sub>E</sub>X* Collection.

I plan on putting out an update to the DECUS *T<sub>E</sub>X* Collection in August. I am currently looking for any submissions that would be helpful in the DEC computing environment (not necessarily on DEC computers).

Support for the following operating systems will be available:

- VMS
- Ultrix/Unix
- MS-DOS
- Macintosh
- Amiga-DOS

I am also planning to put out an Ultrix/Unix version of the tape. It would still have everything the other tape has, only in Ultrix (instead of VMS) biased format.

I have a couple of people helping me this time around, so I hope to be able to do more.

Also, there will be a number of *T<sub>E</sub>X*-related items on the Spring 1990 L&T SIG tape in the EPUBS subdirectory. Included will be the latest XDVI for DECWindows (and it is very nice), the update to GPLOT/GTEX, an update to DVIOUT, a document that is an introduction to *T<sub>E</sub>X*, *T<sub>E</sub>X* examples, and updates to the *T<sub>E</sub>X* help files for VMS. I am still looking at a few other items that may make the spring tape, but my major concern is for the actual *T<sub>E</sub>X* tape for the fall.

Anyone with something to offer is invited to send me a description, preferably by electronic mail.

◊ Ted Nieland  
 Control Data Corporation  
 Suite 200  
 2970 Presidential Drive  
 Fairborn, OH 45324  
 TNIELAND@AAMRL.AF.MIL

## Q & A

### JUST PLAIN Q&A: New Column

Alan Hoenig

### Look Here for TeX Advice

The Editorial Staff at *TUGboat* is planning a new column. If you let us know your TeX problems and puzzles, we'll answer them in this column. This column *can't* fly without reader response, so please write!

We hope to provide a service comparable to that available on the TeXhax network. That an ever increasing proportion of TUG's members have no access to this network is an important reason for starting this column.

**Ground Rules.** Being naturally optimistic, we expect to receive many more queries than we can possibly deal with in this column, so we'll choose problems whose solution might be particularly instructive to the TeX community at large. Note, though, we solicit only just plain TeX questions. Continue to forward L<sup>A</sup>T<sub>E</sub>X problems to Jackie Damrau, and she will continue to ably provide assistance in that area.

No problem is too trivial or elementary! As a reliable rule of thumb, assume that if you have a problem, so do (dozens of) others. If we use your question, we'll be pleased to include your name and affiliation in the column (although we will honor requests for anonymity from the modest and humble).

Try to keep your questions specific. We're not enthusiastic about answering questions like *how would you design a complete macro package to typeset a newsletter?*

Write or phone your problems to the undersigned. (But if you phone, please realize that we can't and won't provide an answer on the spot.) You may e-mail your problems directly to *TUGboat* at *TUGboat@Math.AMS.com*. Please time your inquiries so we receive them at least **four weeks** prior to the current *TUGboat* submission deadline. (You can find this calendar inside your current *TUGboat*.) Items received after a deadline will be considered for the next issue.

We hope to hear from you.

- ◊ Alan Hoenig  
17 Bay Avenue  
Huntington, NY 11743  
(516) 385-0736

## Tutorials

### Output Routines: Examples and Techniques.

#### Part II: OTR Techniques.

David Salomon

The warnings and disclaimers in Part I\* of this article also apply to this part. The methods and macros described here are not canned. They should not be copied and used verbatim. Rather, they should be carefully studied and adapted to specific needs.

The following techniques are discussed in this article, and are applied to practical situations:

1. Breaking up \box255 in the OTR into individual lines by means of the \lastxx commands.
2. Identifying individual lines or paragraphs to the OTR by means of \rightskip, \parshape, or the depth of \box255.
3. Attaching very small amounts of \kern to certain lines of text, to identify those lines to the OTR as special.
4. Placing large negative penalties at certain points in the document. This has the effect of invoking the OTR at those points. The OTR does not have to shipout anything.
5. Attaching very small vboxes below certain lines, to identify them to the OTR as special lines that require special treatment.
6. Using marks. This is a common OTR technique.
7. Setting \vsize to a very small value. \box255 consists, in such a case, of just one line of text, which is then easy to examine.
8. Using a 2-pass technique where, in the first pass, certain information is written on a file, to be read by the second pass. Certain complex problems may even call for a multi-pass job.

We also remind the reader of the notation used in Part I: [...] alone makes reference to an item or items in *The TeXbook* (e.g., [400] refers to page 400 and [Ch. 6] refers to Chapter 6 in *The TeXbook*), whereas [§...] refers to a module or modules in *TeX: The Program*.

---

\* *TUGboat* 11, no. 1, pp. 69–85.

### Technique: Special Penalties

Penalties are used in TeX to control line breaks and page breaks, depending on the current mode. Penalties generated in h-mode are used by the paragraph break algorithm [§831, §859]. To communicate with the OTR by penalties, they therefore have to be generated in v-mode. A penalty of 10000 or more is considered infinite and prevents a page break. Similarly, a penalty of -10000 or less always causes a break. The idea is to say `\penalty-10001` at any point that requires the OTR's attention (TeX must be in v-mode at that time), in order to invoke the OTR at this point. A macro such as

```
\def\immed{\vadjust{\penalty-10001}}
```

can be used for this purpose. The OTR should check `\outputpenalty` and, if it equals -10001, do something special. It can then shipout `\box255` or return it to the current page.

This is a good method for communicating with the OTR, and has only one feature that makes it less than ideal; the special penalty value of -10001 does not invoke the OTR *immediately*. Instead, it is initially placed in the recent contributions, together with the rest of the paragraph, and has to wait until the page builder is exercised. The problem is that, when the page builder is exercised and the OTR invoked, TeX has already read text past the special penalty.

In a test such as

```
..\dimen0=1pt...\immed...\dimen0=2pt..\par
```

the OTR would find `\dimen0` to have a value of 2pt.

**Exercise:** Write an OTR that displays the value of `\dimen0`, and perform the test above.

The reason for this behaviour is the way `\vadjust` is executed. TeX first breaks the entire paragraph into lines that are placed in the recent contributions. Only then does it place the `\vadjust` material at the proper point between two lines [259]. As a result, the OTR is invoked too late.

To solve this problem, a way should be found to exercise the page builder immediately. The page builder is exercised (see [117]) at the start and end of a paragraph; so, if the user wants to invoke the OTR at the end of a paragraph, a `\penalty-10001` is the ideal technique. The page builder is also exercised before and after a display formula, which suggests a way to exercise it inside a paragraph. The user should place, in the paragraph, a `\penalty-10001`, preceded by an empty display formula, at the point where the OTR should be invoked.

An empty formula is easy to create by `$$ $$`. Furthermore, the large flexible glues surrounding a display are easily eliminated by:

```
\abovedisplayskip=1sp  
\belowdisplayskip=1sp  
\abovedisplayshortskip=1sp  
\belowdisplayshortskip=1sp
```

To eliminate any extra interline spaces around the display, an `\openup-\baselineskip` is placed in it. Finally, setting `\postdisplaypenalty=-10001` places the special penalty right below the display formula, to make sure that the OTR is invoked.

The result is made into a new definition of macro `\immed`:

```
\def\immed{${}\postdisplaypenalty=-10001  
\openup-\baselineskip$}
```

The expansion `\immed` terminates the current line (same as `\hfil\break`), places an empty, invisible display formula following the line, and *immediately* invokes the OTR with `\outputpenalty = -10001`. The paragraph is not terminated.

To see the point where the formula is placed, `\immed` can be temporarily changed to:

```
\def\immed{${}\postdisplaypenalty=-10001  
\openup-\baselineskip+$}
```

In a test such as

```
..\dimen0=1pt...\immed...\dimen0=2pt..\par
```

the OTR would find `\dimen0` to have a value of 1pt.

This method is, again, not ideal, since it terminates the current line.

### The `\lastxx` Commands

The OTR can examine the contents of `\box255` and also break it up into its components, by means of the `\lastxx` commands [§424, §996]. There are 4 of them: `\lastbox`, `\lastskip`, `\lastkern` and `\lastpenalty` [271]. To use those commands, the OTR should first open `\box255`, by means of an `\unvbox`. If the last item in `\box255` is a glue, its value will be reflected in `\lastskip`. Two things can be done at this point (1) `\skip0=\lastskip`; (2) `\unskip`. The first saves the glue value for future use, and the second removes it [280]. Similarly for `\lastkern` and `\lastpenalty`. If the last item is a box, the command `\setbox0=\lastbox` will both set `\box0` and remove the last box.

### Technique: Breaking Up a Page

The OTR may use the `\lastxx` commands in a loop, to identify successive components of `\box255`. In such a loop it is, of course, important to check at

each iteration and find out what the next item is, before copying and removing it. If the next item is not a glue, `\lastskip` will have a value of `Opt`. Similarly, `\lastkern` will be `Opt`, `\lastpenalty` will be 0, and `\lastbox` will be `void`. A macro `\breakup` can thus be defined, consisting of a `\loop... \repeat` to remove successive elements off `\box255`.

```
\newif\ifAnyleft \newcount\pen
\def\breakup{%
  \loop \Anyleftfalse
    \ifdim\lastskip=Opt\else \Anylefttrue
      \skip0=\lastskip \unskip \fi
}
```

---

```
\newif\ifAnyleft
\def\breakup{%
  \loop \Anyleftfalse
    \ifdim\lastskip=Opt \ifdim\lastkern=Opt \ifnum\lastpenalty=0
      \setbox0=\lastbox \ifvoid0      % end of breakup loop
      \else \Anylefttrue \fi        % box encountered
      \else \Anylefttrue \unpenalty \fi % penalty encountered
      \else \Anylefttrue \unkern \fi   % kern encountered
      \else \Anylefttrue \unskip \fi   % glue encountered
    \ifAnyleft \repeat}
```

Before discussing specific applications of the breakup technique, let's look at its main problems.

1. We have to test `\lastskip` for `Opt`. Unfortunately, TeX does not have an `\ifskip` or `\ifglue` tests. We thus have to use `\ifdim`, which tests a dimension, not a glue. The test `\ifdim\lastskip...` first converts the glue to a dimension. The problem is that such a conversion discards the stretch and shrink components of the glue [118]. Thus if the next glue item has the form `Opt plus.. minus..`, our macro will consider it zero.

The solution: change the values of certain common vertical glues that have this form to `1sp plus... minus....` We thus declare:

```
\parskip=1sp plus1pt
\def\vfil{\vskip1sp plus1fil}
\def\vfille{\vskip1sp plus1fill}
\abovedisplayshortskip=1sp plus3pt
```

2. A similar problem exists with penalties. A math display formula is followed by a `\postdisplaypenalty` [189], whose default value is zero. As a result, any construct using the math display mode, such as `\verbatim` or `$$\vbox{\halign{...}}$$`, suffers from the same problem. The solution is to set `\postdisplaypenalty=1`.

```
\ifdim\lastkern=Opt\else \Anylefttrue
  \dimen0=\lastkern \unkern \fi
\ifnum\lastpenalty=0 \else \Anylefttrue
  \pen=\lastpenalty \unpenalty \fi
\setbox0=\lastbox
\ifvoid0 \else \Anylefttrue \fi
\ifAnyleft \repeat}
```

Note the use of variable `\Anyleft` to check if there is anything left in the box after each repetition of the loop. The loop repeats until none of the four items is found. The OTR simply says `\unvcopy255 \breakup`.

An alternative definition of `\breakup`, using nested `\ifs`, is:

There is also an `\interlinepenalty` parameter, which goes between the lines of a paragraph. It is usually zero but can be changed to a large value [406] to discourage a page break inside a paragraph. We set it to 1.

The above definitions are all consolidated into a new macro `\zeroToSp`, which should be used in conjunction with any page breakup.

```
\def\zeroToSp{\parskip=1sp plus1pt
\def\vfil{\vskip1sp plus1fil}
\def\vfille{\vskip1sp plus1fill}
\abovedisplayshortskip=1sp plus3pt
\postdisplaypenalty=1
\interlinepenalty=1}
```

3. When breaking up a box using the `\lastxx` commands, it is easy to identify the 4 types: box, glue, kern and penalty. There seems no way, however, to identify the other three components of vertical lists, namely *rules*, *marks* and *whatsits*. When our breakup loop gets to one of them, it stops, assuming that this is the end of `\box255`. A whatsit (a `\special` or a `\write`) can usually be specified in horizontal mode, which will bury it inside an `\hbox` and out of harm. A mark, on the other hand, tends to migrate outside horizontal lists [400] and into the top level of `\box255`. It therefore

causes an incomplete breakup, and its use should be avoided when this technique is employed.

A similar problem is presented by a rule. An `\hrule` at the top level of a `\vbox` is considered a box [110]. However, the `\lastbox` operation cannot identify it as such, which results in an incomplete breakup.

A solution: Place the `\hrule` in its own `\vbox`, so it does not appear at the top level of the larger `\vbox`.

Partial relief: Such a case, where the breakup stops prematurely, can be detected by setting a new box (`\brk`) to the remains of `\box255` after the breakup. When the breakup stops, `\ht\brk` should be zero. An OTR can thus be written which breaks up a copy of `\box255`, and checks to see if anything is left.

```
\zeroToSp
\newif\ifAnyleft \newcount\pen
\def\duplicate{%
\loop \Anyleftfalse
\ifdim\lastskip=0pt \ifdim\lastkern=0pt \ifnum\lastpenalty=0
    \global\setbox0=\lastbox \ifvoid0 % end of breakup loop
    % box present
\else \Anylefttrue
    \global\setbox1=\vbox{\box0 \unvbox1}\fi
\else \Anylefttrue
    \pen=\lastpenalty
    \global\setbox1=\vbox{\penalty\pen\unvbox1}\unpenalty\fi
\else \Anylefttrue
    \dimen0=\lastkern
    \global\setbox1=\vbox{\kern\dimen0 \unvbox1}\unkern\fi
\else \Anylefttrue
    \skip0=\lastskip
    \global\setbox1=\vbox{\vskip\skip0 \unvbox1}\unskip\fi
\ifAnyleft \repeat}
```

A test such as:

```
\newbox\brk
\output={%
\setbox\brk=\vbox{\unvcopy255 \duplicate}
\ifdim\ht\brk>0pt
    \message{Incomplete breakup}\fi
\shipout\box255 \shipout\box1
\advancepageno}
```

is particularly interesting. It typesets pairs of pages, with the same page numbers. Two *physical pages* are printed for each *logical page* generated. The two pages in a pair are duplicates of each other, but are they identical?

```
\newbox\brk
\output={%
\setbox\brk=\vbox{\unvcopy255 \breakup}
\ifdim\ht\brk>0pt
    \message{Incomplete breakup}\fi
\shipout\box255 \advancepageno}
```

**Exercise:** Implement the above OTR and use it to typeset several pages, some containing rules or marks.

Here are a few simple applications of the breakup technique.

### Duplicating a Page

Macro `\breakup` can be modified to place broken up components from `\box255` in `\box1` in the *original order*, creating a copy of the current page.

It turns out that they are not. The main difference between `\box255` and `\box1` is their heights. The heights are different because of the flexible glues on the page. Normally, `\box255` contains some flexible vertical glues. Those glues are flexed to adjust `\ht255` to equal `\vsize`. When `\box255` is opened, however, *the glues return to their natural size*.

This can easily be seen by a test such as:

```
\newbox\brk
\output={%
\setbox\brk=\vbox{\unvcopy255 \duplicate}
\ifdim\ht\brk>0pt
    \message{Incomplete breakup}\fi
\message{[\the\ht255:\the\ht1]}
```

```
\shipout\box255 \shipout\box1
\advancepageno}

\parskip=6pt plus6pt minus6pt
\input source
\bye
```

The `\parskip` glue is given a lot of flexibility, and the heights are shown in the log file. Such a test also shows that the last pair of pages may differ a lot in their heights. This is because the last page of a document is normally only partly full, and has a `\vfill` glue at the bottom. When `\box255` is opened, the `\vfill` returns to its natural size, which is Opt.

How can we make sure that the two pages in a pair have the same heights? The simplest approach is to flex `\box1` in the OTR, just before it is shipped out, by saying `\setbox1=\vbox to\vsizet\unvbox1}`. Now the two pages in a pair have exactly the same height and the same glue set ratio; they are identical. Our OTR thus becomes:

```
\newbox\brk
\output={
  \setbox\brk=\vbox{\unvcopy255 \breakup}
  \ifdim\ht\brk>0pt
    \message{Incomplete breakup}\fi
  \setbox1=\vbox to\vsizet\unvbox1
  \shipout\box255 \shipout\box1
  \advancepageno}
```

Two `\showbox` commands can temporarily be placed in the OTR to dump `\box1` and `\box255` onto the log file, and verify that they have identical components. It is important to (temporarily) increase the value of `\showboxbreadth`. Also, to make the dumps more manageable, `\vsizet` should be set to a small value, such as 1in.

## Reversing a Page

It is now trivial to modify the definition of `\duplicate`, so that it breaks up items from `\box255` and places them in `\box1` *in reverse order*. This is, perhaps, a useless operation but, since our aim is to gain an understanding of OTRS, let's ask ourselves how `\box255` and `\box1` differ.

1. They are the reverse of each other, which means that each glob of `\baselineskip` glue which used to be below a line of text, is now above it. The interline spacing in `\box1` is thus all wrong. This is not very noticeable when the entire page is typeset with the same font. Mixing different font sizes, however, results in a funny looking reversed page. Also, the `\parskip` glues are misplaced but, since they are normally zero, this is not noticeable. Changing `\parskip` to some non-zero value results in large spaces following the first line of each paragraph (which are last lines on the reversed page).

2. They have different vertical dimensions. The height of `\box255` is `\vsizet` and its depth is usually the depth of the last line of text. `\box1`, on the other hand, ends with `\topskip`, which is glue and thus has no depth, so `\dp1=0pt`. Also, `\box1` starts with the bottom line of `\box255`. To guarantee that `\ht1+\dp1` equals `\ht255+\dp255`, we should force `\ht1` to be the sum `\ht255+\dp255`.

**Exercise:** Write a macro `\reversepage` to reverse `\box255` into `\box1`.

## Counting the Lines

The `\breakup` macro can now easily be modified to count the number of lines of text in `\box255`. We assume that `\box255` does not contain rules, marks or whatsits, and we break it up, counting the number of `\hbox`s found. Items that we don't want to count should be placed in a `\vbox`. The macros are:

---

```
\zeroToSp
\newif\ifAnyleft \newcount\lineCount
\def\countlines{\global\lineCount=0
  \loop \Anyleftfalse
    \ifdim\lastskip=Opt \ifdim\lastkern=Opt \ifnum\lastpenalty=0
      \setbox0=\lastbox \ifvoid0
        \else \Anylefttrue \ifhbox0 \global\advance\lineCount by1 \fi \fi
        \else \Anylefttrue \unpenalty \fi
        \else \Anylefttrue \unkern \fi
        \else \Anylefttrue \unskip \fi
      \ifAnyleft \repeat}
\newbox\brk
```

```
\output={\setbox\brk=\vbox{\unvcopy255 \countlines}
        \ifdim\ht\brk>0pt \message{Incomplete breakup} \fi
        \message{\the\lineCount}
        \shipout\box255 \advancepageno }
```

### Breaking Up a Line of Text

Can we use the same technique to break up individual lines of `\box255`? It seems easy to define a macro `\hbreakup` that would use `\lastxx` commands to break up a line of text. Unfortunately, this does not work, because a line of text contains individual characters, which the `\lastbox` command cannot recognize as boxes. It is interesting to note that a character of text is, in general, considered a box [63] but, evidently, there are differences between a general box and a character box. One such difference is that a character box cannot appear in a vertical list [110]. Another difference is the one mentioned above, concerning `\lastbox`, and this difference is easy to verify with a test such as

```
\setbox0=\hbox{ABC}
\unhbox0 \setbox1=\lastbox
\showbox1
\bye
```

which shows `\box1` to be void, and typesets 'ABC'. In contrast, the test:

```
\setbox0=\hbox{AB\hbox{C}}
\unhbox0 \setbox1=\lastbox
\showbox1
\bye
```

shows `\box1` to consist of an `hbox` with the 'C', and typesets only 'AB'.

This is an unfortunate situation. The ability to break up a line of text would have meant a full and complete communication with the OTR. The user could hide, e.g., a strut with a special depth in the line, and the OTR could easily find it, and do something with, or add something to, the line at that point. The strut could even have been left in the line.

### Technique: Using `\rightskip`

Even though `\lastbox` cannot be used to break up a line of text, `\lastskip` can be used to detect glue at the right end of such a line. This suggests a way to identify certain lines to the OTR. How can a glob of glue be placed at the end of a line? It turns out that TeX places the `\rightskip` glue at the end of every line of text when the paragraph is broken into lines. The plain format value of `\rightskip` is `0pt` so, setting `\rightskip=1sp` will not be

visually noticeable and can be used to communicate with the OTR. Unfortunately "TeX uses the same `\rightskip` value in all lines of a paragraph" [393]. This method can thus be used to identify certain paragraphs, but not individual lines, to the OTR.

An application demonstrating this technique is shown later. It has to do with 'special boxes' in a textbook. Following are two examples that are not developed in detail, since they are easier to do in other ways:

1. Suppose that a vertical rule should be typeset on the left margin of certain paragraphs. The OTR can do this by placing a rule, the size of a strut, on the left of each line that ends with `\rightskip=1sp`. However, this is easier to do by typesetting the paragraph in a `\vbox` and placing a rule on the left of the box.

2. If only one or two lines of the paragraph appear on (the bottom of) the page, we want to move them to the next page, and to `\vfill` up the present one. This can be done by the OTR checking the `\rightskip` glue of the bottom line or two. However, it may be easier to do with `\filbreak` [111].

### Technique: Using `\parshape`

If we want the OTR to do something special with, say, the second line of a paragraph, we can identify this line by making it `1sp` longer or shorter than the other lines. This can easily be done with `\parshape`. Again, there are no practical applications as yet for this technique.

### Technique: Using the Depth of `\box255`

The following quote, from [400], is relevant to this technique: "Perhaps the dirtiest trick of all is to communicate with the OTR via the *depth* of `\box255`." After mastering the techniques described here, the reader will agree that this is no longer the dirtiest trick, but is a special case of the breakup technique. Examples of applications of this technique are:

1. In certain old religious texts, if a chapter ends on a certain page, and less than half a page remains, the next chapter should start on the following page; otherwise, it should start on the same page.

2. Business contracts usually consist of clauses. In certain legal situations it is desirable to break a page between clauses. If the page must be broken inside a clause, a special footer should be typeset, saying *Continued....* This can be done by ending each clause with `\endclause`, a macro defined as `{\unskip\vrule height0pt width0pt depth3.5002pt}`. The `\unskip` backspaces over any possible space preceding the special strut, thus making sure that the strut will end up on the same line as the preceding word.

The OTR simply tests

```
\ifdim\dp255=3.5002pt \else
  \footline={\hfil\sevenrm Continued...}
\fi
```

3. Certain lines should not appear at the bottom of the page. A business contract is again a good example. If a certain line contains the most important words or money sums in the contract, it should better not appear at the bottom of the page, where it is less visible.<sup>†</sup> Again, a special strut can be used to identify the line and, if the OTR detects such a line, it should alert the user, who can then correct the situation by rewording the document, or by moving things around.

#### Technique: Communications by Kerns

Small amounts (a few sp worth) of `\kern` can be placed between the lines of text, and detected by the OTR when breaking up `\box255`. The problem is that a kern is discardable, so we have to make sure that our special kern is not discarded. The general rule is that a page can be broken at a kern only if the kern is immediately followed by glue. We, therefore, will have our special kern followed by another kern. In fact, we will place two consecutive, identical pieces of special, small kern after the text line that we want to identify to the OTR. This is done by `\vadjust{\kern1sp\kern1sp}`, which places the kerns immediately below the current line, i.e., they are placed between the line and the `\baselineskip` that normally follows it. If the line should be followed by a penalty, the order is:

---

<sup>†</sup> Beware! Certain businessmen do just this.

the line of text, the pair of kerns, the penalty, the `\baselineskip`. The places where a page can be broken have been mentioned earlier.

#### Practical Examples of OTRs

The techniques described earlier, plus a few others, are now applied to practical problems.

#### Example: Start a Chapter On a New Page

The problem\*: If a chapter ends on a certain page, and less than half a page remains, skip the rest of the page; otherwise, start the new chapter on the same page.

Solution: Macro `\chapter` is expanded at the start of each chapter. It appends a special line to the end of the preceding chapter (only if there is a preceding chapter), and invokes the OTR by an `\eject`. The special line consists of just a small `\hbox` with a rule of depth `1sp`, and width and height zero.

Each time the OTR is invoked, it checks to see if `\dp255=1sp` and `\ht255<0.5\vsiz`. If yes, the OTR returns `\box255` to the MVL (it is an end-of-chapter and more than half a page remains); otherwise, `\box255` is shipped out (either less than half a page remains or not end-of-chapter).

Actually, the details are a bit more involved. If `\dp255=1sp`, then `\box255` contains text, followed by a `\vfill`, and by the special box. The last two items have to be removed before `\ht255` can be tested. To do this—

1. `\box 255` is opened, the special box at the bottom is removed by a `\lastbox` (see later), and the `\vfill` is skipped over by an `\unskip`. The result then goes back in `\box255`. The new `\box255` now has just the original text, and its height can be measured.

2. If `\ht255<0.5\vsiz`, `\box255` is opened, and a message (unv) goes in the log file. Otherwise, a new box is shipped, consisting of `\box255`, a `\vfill`, and a footnote. The size of the new box is `\vsiz+12pt`, and it has to be explicitly specified.

A listing of the macros follows. They are kept, as usual, simple.

---

\* Proposed by Robert Batzinger.

```
\newif\ifFirstCh \FirstChtrue \newif\ifRet \newcount\chnum
\newdimen\Hvsiz \Hvsiz=\vsiz \divide\Hvsiz 2
\newdimen\Nvsiz \Nvsiz=\vsiz \advance\Nvsiz 12pt
```

```

\def\chapter#1\par{\advance\chnum 1
    \ifFirstCh \FirstChfalse
    \else \vfill\nointerlineskip
        \hbox{\vrule width0in height0pt depth1sp}
        \eject \fi
    \bigskip\noindent{\bf \the\chnum.\ #1}
    \medskip}

\footline={...}

% if \dp255 = 1sp: unvbox255, lastbox (the line with dp = 1sp),
% skip over the \vfill by \unskip, and return to MVL.

\output={\Retfalse
    \ifdim\dp255=1sp
        \setbox255=\vbox{\unvbox255 \setbox0=\lastbox \unskip}
        \ifdim\ht255<\Hvsize \Rettrue \fi \fi
    \ifRet \unvbox255 \message{unv}
    \else
        \shipout\vbox to\Nvsize{\box255\vfill\line{\the\footline}}
        \advancepageno \message{ship} \fi
    }
}

```

### Example: A Religious Hymn

One way of communicating with the OTR, proposed in [App. D], is by the use of special penalty values. Any penalty value  $\leq -10000$  will cause the OTR to be invoked. Values  $< -10000$  can therefore be used to tell the OTR to do something special.

Note that the OTR is not invoked when *TEX* first sees the penalty. It is only invoked when the page builder detects the penalty, while moving items from the recent contributions to the current page [§1005].

The OTR should check the value of variable *\outputpenalty*. If it is  $< -10000$ , it should do something special and then return *\box255* to the MVL without shipping out anything (a dead cycle). If, however, *\outputpenalty* equals  $-10000$ , the OTR should do a normal *\shipout*.

The example shown here\* has to do with typesetting a religious hymn. A hymn consists of one chorus and a number of stanzas. The chorus is usually printed after the first stanza and is sung

---

\* Proposed and solved by Robert Batzinger.

---

after each stanza. The problem is that a long hymn may occupy more than one page and, in such a case, the chorus should be printed on the top of each successive page.

The solution is to write macros that will typeset a copy of the chorus if we are still within the same hymn, but have moved to a new page. The original text of the chorus is saved in a *\toks* variable, so it can be used as often as necessary.

Macro *\hymn* is expanded at the start of each hymn. It invokes the OTR with penalty  $-10001$  and the OTR, in that case, simply saves the current page number in the count variable *\oldpage*. Note that the OTR does not shipout anything, and returns *\box255* to the MVL.

Macro *\stanza* is expanded at the start of each stanza. It invokes the OTR with penalty  $-10002$ . The OTR then tests *\ifnum\pageno>\oldpage* (we have moved a page or two since the last printing of the chorus) and sets the boolean variable *\prtCorus* to true. The OTR then returns *\box255* to the MVL. Macro *\stanza* tests *\prtCorus* and, if it is true, invokes macro *\setchorus* to typeset the chorus.

Here are the macros used:

```

\newif\ifprtCorus \newcount\oldpage
\hsize=3.5in \vsize=2.2in

\def\hymn#1#2{\bigbreak\bigskip
    \noindent{\bf #1. #2}\nobreak\medskip
    \nobreak \penalty-10001}

```

```

\def\stanza#1\endstanza{\medbreak
    \vbox{\noindent#1}\medskip\penalty-10002
    \ifprtCorus \setchorus\fi }

\def\chorus#1\endchorus{\toks2={#1}\setchorus}

\def\setchorus{\medskip
    \moveright.5in\vbox{\noindent
        \hbox to 0pt{\hss\bf Chorus:\ }%
        \the\toks2\medskip}
    \global\prtCorusfalse }

\output={\ifnum\outputpenalty=-10001
    \global\oldpage=\pageno
    \global\prtCorusfalse
    \unvbox255
\else \ifnum\outputpenalty=-10002
    \ifnum\pageno>\oldpage
        \global\prtCorustrue \global\oldpage=\pageno \fi
    \unvbox255
\else
    \shipout\vbox{\box255\smallskip \line{\the\footline}}
    \advancepageno
\fi \fi }

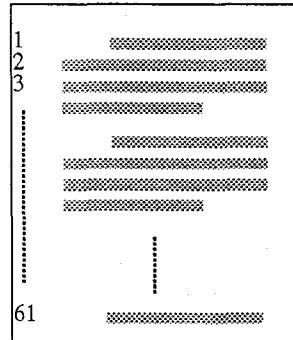
```

Note that this is just a demonstration of a principle. The macros presented here are simple and will not always work. One case where they fail is when a hymn starts at the end of a page, and the chorus is typeset on the following page. The chorus will, in such a case, be typeset twice on that page. There may be other problems, but the idea in this article is to keep the macros simple and easy to read.

**Exercise:** Generalize the above macros so that they do not typeset the chorus on an odd-numbered (right hand) page if it was typeset on the preceding even-numbered (left hand) page. This way the chorus would be typeset only once on a pair of facing pages.

#### Example: Line Numbering

When writing a draft of a book, a thesis, or a report that should be reviewed by someone else, it is useful to number the lines on each page (see Figure 1). This way the reviewer can easily refer to, e.g., *line 48, page 84*. The numbers should be placed in the margin, so they can be suppressed in the final version without any changes in the layout of the document.



Line Numbers on the Margin  
Figure 1

The method used here counts the number of lines of text by counting the boxes that make up the page. Macro `\countlines` below assumes that each box on the page is a line of text and should be numbered. Alternatively, if certain items on the page should not be numbered, they can be placed in vboxes, and `\countlines` revised to count only hboxes.

```
\zeroToSp
\newif\ifAnyleft \newcount\lineCount
\def\countlines{%
  \global\lineCount=0
  \loop \Anyleftfalse
    \ifdim\lastskip=0pt \ifdim\lastkern=0pt \ifnum\lastpenalty=0
      \setbox0=\lastbox \ifvoid0
    \else \Anylefttrue \global\advance\lineCount by 1 \fi
    \else \Anylefttrue \unpenalty \fi
    \else \Anylefttrue \unkern \fi
    \else \Anylefttrue \unskip \fi
  \ifAnyleft \repeat}
```

Note that, in an `\halign`, each line becomes an `\hbox` and is, therefore, counted separately. Also note that a blank line preceding a display equation becomes an empty paragraph, and is therefore counted.

The OTR breaks up a copy of the page, removing the lines of text one by one. At the same time, a new box, `\box1`, is built, from the bottom up,

```
\newcount\pen
\def\breakup{%
  \loop \Anyleftfalse
    \ifdim\lastskip=0pt \ifdim\lastkern=0pt \ifnum\lastpenalty=0
      \global\setbox0=\lastbox \ifvoid0 % end of breakup loop
    \else \Anylefttrue \appendline \fi
    \else \Anylefttrue \pen=\lastpenalty
      \global\setbox1=\vbox{\pen \unvbox1} \unpenalty \fi
    \else \Anylefttrue \dimen0=\lastkern
      \global\setbox1=\vbox{\kern\dimen0 \unvbox1} \unkern \fi
    \else \Anylefttrue \skip0=\lastskip
      \global\setbox1=\vbox{\vskip\skip0 \unvbox1} \unskip \fi
  \ifAnyleft \repeat}

\def\appendline{%
  \setbox2=\hbox{\vrule height\ht0 depth\dp0 width0pt \sevenrm\the\lineCount}
  \global\advance\lineCount-1
  \global\setbox1=\vbox{\box2 \unvbox1}

\newbox\brk
\output={\global\lineCount=0
  \setbox\brk=\vbox{\unvcopy255 \countlines}
  \global\setbox1=\vbox{}
  \setbox\brk=\vbox{\unvcopy255 \breakup}
  \ifdim\ht\brk>0pt \message{Incomplete breakup} \fi
  \ht1=0pt \dp1=0pt
  \shipout\vbox{\moveleft20pt\box1 \box255}
  \advancemodepage}
```

with the line numbers on the margin. For each line removed from the page, its height and depth are measured, and a line with the same size, containing the appropriate number, is added to the top of `\box1`. Each glue or kern removed from the bottom of the page is added to the top of `\box1`. At the end, the height and depth of `\box1` are set to zero and it is typeset, superimposed on the original page.

This example illustrates both the power of the breakup technique, and its main problem. The problem is the flexible glues in `\box255`. They are flexed, by the page builder [§668, §1017], to adjust `\ht255` to `\vsize`. However, when `\box255` is opened, for the breakup, the flexible glues return to their natural size.

A partial solution is to reduce, or even eliminate, the flexibility of those glues (mainly `\parskip`). This, however, handicaps the page builder in its most important task, namely, finding a good point to break a page.

**Exercise:** Implement an alternative approach to the line numbering problem. The new approach should build, in `\box1`, a duplicate of `\box255` with the line numbers inserted on the left.

#### Example: Special Footnote Numbering

Another practical problem\* is to number the footnotes in a document by the line number on the page. This problem is solved here several times, using different approaches. Each approach illustrates

---

\* Proposed by Lothar Meyer-Lerbs.

---

different OTR techniques, and also involves certain difficulties.

The following quote, from the Chicago Manual of Style (see also [125]), is relevant. "Since it is impossible to foresee how [footnotes] will happen to come out in the make-up, it is impracticable to number them from 1 up on each page. The best way is to number them consecutively throughout an article or by chapters in a book." The problem tackled here is much more complicated than the one proposed in the quote, and demonstrates the power of OTRs in `TEX`.

#### A Simple, Wrong Approach

The first approach is simple and intuitive. Macro `\Nfootnote` uses a penalty of `-10001` to invoke the OTR prematurely. The macro is expanded from h-mode, and it has to place the penalty at the top level of `\box255`, between lines of text. This is done with `\vadjust`. The OTR breaks up `\copy255` and counts the number of lines in the page so far. It then returns `\box255` to the MVL. Macro `\Nfootnote` again takes over and typesets the footnote with the number calculated by the OTR.

The macros are very simple:

```
\def\Nfootnote#1{%
  \vadjust{\penalty-10001}%
  \footnote{$^{\the\lineCount}\{#1\}}}

\zeroToSp
\newbox\brk \newif\ifAnyleft \newcount\lineCount

\def\breakup{%
  \global\lineCount=0
  \loop \Anyleftfalse
    \ifdim\lastskip=0pt \ifdim\lastkern=0pt \ifnum\lastpenalty=0
      \global\setbox0=\lastbox \ifvoid0
      \else \Anylefttrue \ifhbox0 \global\advance\lineCount1 \fi \fi
      \else \Anylefttrue \unpenalty \fi
      \else \Anylefttrue \unkern \fi
      \else \Anylefttrue \unskip \fi
    \ifAnyleft \repeat}

\output={\ifnum\outputpenalty=-10001
  \setbox\brk=\vbox{\unvcopy255 \breakup}
  \ifdim\ht\brk>0pt \message{Incomplete breakup} \fi
  \unvbox255
  \else \plainoutput \fi
}
```

but they don't work! The serious reader should, by now, know the reason. The `\vadjust` with the special penalty does not invoke the OTR *immediately*. Instead, the penalty is placed following the current line. Thus, in the second part of `\Nfootnote`, when

it expands `\footnote`, the OTR has not yet been invoked.

### A 2-pass Method

The idea in the second approach is to modify the OTR so that it writes `\the\lineCount` on a file. This leads to a 2-pass job, shown below.

```

\newcount\lineCount \newbox\brk \newbox\sav \newcount\pass

\newread\aux \immediate\openin\aux=\jobname.lin
\ifeof\aux \immediate\openout\aux=\jobname.lin \pass=1 \message{pass 1}
\else \pass=2 \message{pass 2} \fi

\newif\ifAnyleft
\zeroToSp

\def\countlines{%
\global\lineCount=0
\loop \Anyleftfalse
\ifdim\lastskip=0pt \ifdim\lastkern=0pt \ifnum\lastpenalty=0
\global\setbox0=\lastbox \ifvoid0
\else \Anylefttrue \global\advance\lineCount by1 \fi
\else \Anylefttrue \unpenalty \fi
\else \Anylefttrue \unkern \fi
\else \Anylefttrue \unskip \fi
\ifAnyleft \repeat}

\output={\ifnum\outputpenalty=-10001
\ifnum\pass=1
\setbox\brk=\vbox{\unvcopy255 \countlines}
\ifdim\ht\brk>0pt \message{Incomplete breakup} \fi
\immediate\write\aux{\the\lineCount} \fi
\unvbox255 % return to MVL
\else \plainoutput \fi
}
% shipout with footnotes

\def\Nfootnote#1{%
\ifnum\pass=1 \vadjust{\penalty-10001}\footnote*{#1}%
\else \read\aux to\tmp \footnote{$^{\tmp}{#1} \fi}%
}
```

In the first pass, macro `\Nfootnote` creates the special penalty and also expands `\footnote*{...}` to typeset the footnote, so it takes the right amount of space on the page. In the second pass, the macro reads the correct number off the file, and invokes `\footnote` with that number. This is still simple and usually works.

It may fail, however, in cases where a footnote appears close to the bottom of the page. Imagine a footnote on line 60 of page 4. Because of the

`\penalty-10001` following this line, `TEX` will invoke the OTR with a 60-line page. The OTR will (1) write the line count, 60, on the file; (2) return `\box255` (with the 60 lines) to the current page, removing the special penalty. Since the current page is now large, `TEX` immediately starts looking for a good page break. It may decide, since the special penalty isn't there any more, to break the page after line 59. Line 60 thus becomes line 1 of the next page,

but the number 60 has already been written on the file.

### Another 2-pass Solution

The third approach is similar except that, instead of being written on a file, the line numbers are saved—by the OTR—in memory. This makes sense since there usually aren't many footnotes on any single page. In the second pass, macro `\Nfootnote` uses this information to expand `\footnote` with the correct line numbers. This approach suffers from the same problem as the previous one, but

it is shown here because it illustrates how to save the line numbers, each as an `\hbox`, in a large `\vbox`. Extracting them later is easily done with a `\lastbox`.

Note that the 2-pass structure is different from the previous one. Previously, each pass was a separate TeX job, and the line numbers were saved on a file between the jobs. In the present method, however, the line numbers are saved in a box (`\sav`), which is stored in memory and thus disappears at the end of the job. The two passes must, therefore, be done in the same job. This is faster but requires the source text to be `\input` from a separate file.

---

```

\newcount\lineCount \newbox\brk \newbox\sav \newif\ifAnyleft
\zeroToSp

\def\breakup{%
\global\lineCount=0
\loop\Anyleftfalse
\ifdim\lastskip=0pt \ifdim\lastkern=0pt \ifnum\lastpenalty=0
\global\setbox0=\lastbox \ifvoid0
\else\Anylefttrue \ifhbox0 \global\advance\lineCount1 \fi \fi
\else\Anylefttrue \unpenalty \fi
\else\Anylefttrue \unkern \fi
\else\Anylefttrue \unskip \fi
\ifAnyleft \repeat}

% pass 1
\output={\ifnum\outputpenalty=-10001
\setbox\brk=\vbox{\unvcopy255 \breakup}
\ifdim\ht\brk>0pt \message{Incomplete breakup} \fi
\global\setbox\sav=\vbox{\hbox{\sevenrm the\lineCount}\unvbox\sav}
\unvbox255
\else\plainoutput \fi
}
% The above line should later be changed to:
% \setbox0=\box255 \deadcycles=0,
% since we don't really want to shipout pages in pass 1.

\def\Nfootnote#1{%
\ vadjust{\penalty-10001}%
\footnote*{#1}%

\input source \vfill\eject \pageno=1

% pass 2
\output={\ifnum\outputpenalty=-10001
\unvbox255
\else\plainoutput \fi
}
```

```
\def\Nfootnote#1{%
  \setbox\sav=\vbox{\unvbox\sav \global\setbox0=\lastbox}%
  \footnote{\raise4pt\copy0}{#1}}}

\input source
```

### A Complex, 3-pass Approach

Approach 4: A three-pass job. The first pass determines the line numbers (throughout the document) of lines with footnotes. Those numbers are saved in a `\vbox` called `\Asav`. The second pass counts the number of lines on each page. Those numbers are also saved, in another box, `\Bsav`. The third pass uses the numbers from the two boxes to determine the correct line numbers and to typeset the footnotes. This is complex and, perhaps, can be done in a simpler way. Nevertheless, it has the advantage of demonstrating several useful OTR techniques.

Before describing the 3 passes in detail, here is a simple numeric example: Let's assume that we have three pages, with 50, 30 and 40 lines respectively. There are footnotes on lines 3, 15, 15 and 44 of the first page, and lines, 25 and 34 of the third page. Pass 1 will save the numbers 3, 15, 15, 44, 105 and 114 in `\Asav` (note that 15 occurs twice). In pass 2, the line counts 50, 30 and 40, of the 3 pages are saved in `\Bsav`. Pass 3 starts by extracting the 50 from `\Bsav`. The first 4 times macro `\Nfootnote` is expanded, it extracts the numbers 3, 15, 15 and 44 from `\Asav`. Those

numbers are  $\leq 50$ , so they are used for numbering the first 4 footnotes. The fifth expansion extracts 105 from `\Asav`. This is  $> 50$ , so the next number, 30, is extracted from `\Bsav` and added to the 50. The current footnote number, 105, is still  $> 80$ , so the next number, 40, is extracted from `\Bsav` and added to the 80. The current footnote number, 105, is now  $\leq 120$ , so 80 is subtracted and the result, 25, is used. The last number is 114, again  $\leq 120$ , so again 80 is subtracted, yielding 34.

The steps in each pass are:

**Pass 1.** Macro `\Nfootnote` computes a running number for each footnote, and creates a `\mark` with that number. The footnote itself is not typeset, but `\Nfootnote` typesets an asterisk to occupy space on the line, approximately equal to that taken by the final footnote number. `\vsize` is set to a small value, so the OTR receives a `\box255` with just one line [400], which makes it easy to number the lines throughout the document. Each time the OTR is invoked, it checks `\firstmark`, `\botmark` and compares them to `\topmark`. This way it knows if there are any footnotes on the line. If there are any, the line number is saved in box `\Asav`, once for each footnote on the line.

---

```

1. \newcount\temp \newcount\footno \newcount\lineno \newbox\Asav
2.
3. \def\Nfootnote#1{\advance\footno 1 \mark{\the\footno}*} % typeset an *
4.
5. \output={\global\advance\lineno 1
6.           \temp=\botmark
7.           \advance\temp -\firstmark
8.           \advance\temp 1
9.           \ifnum\firstmark\botmark \ifnum\topmark\firstmark \temp=0 \fi \fi
10.          \% \temp is now the number of footnotes on the current page (one line)
11.           \ifnum\temp>0
12.             \loop
13.               \global\setbox\Asav=\vbox{\vskip\lineno sp \null\unvbox\Asav}
14.               \advance\temp-1
15.               \ifnum\temp>0 \repeat
16.             \fi
17.             \setbox0=\box255 % get rid of
18.             \deadcycles=0
19.           }
20.
```

```

21. % *** Executable commands ***
22. \message{Pass 1;} \vsize=10pt % small value
23. \footno=0 \lineno=0 \setbox\Asav=\vbox{}
24. \input source \eject

```

Lines 1-19 are macro definitions, and declarations of variables. Lines 22-24 are the actual commands executed in pass 1.

\vsize is set, on line 22, to the small value 10pt. The page in \box255 will, as a result, consist of just one line of text.

The OTR calculates \temp, on lines 6-8, as \botmark - \firstmark + 1. \temp is now the number of footnotes on the current page (which consists of just one line of text). However, if

\botmark = \firstmark = \topmark, there are no footnotes on the current line, and \temp is set, on line 9, to 0.

If \temp ≠ 0, the loop, on lines 12-15, saves variable \lineno on top of \box\Asav as glue (in units of sp).

**Pass 2.** Macro \Nfootnote typesets each footnote with an asterisk. No marks are used. \vsize is set to its normal value, and the OTR breaks up a copy of each page, counts the number of lines, and saves that number, as the top glue item, in box \Bsav.

```

1. \newif\ifAnyleft \newbox\Bsav \newbox\brk
2.
3. \def\Nfootnote#1{\footnote*{#1}}
4. % typeset the footnote so it occupies the right space on the page
5.
6. \def\countlines{%
7.   \global\lineno=0
8.   \loop \Anyleftfalse
9.     \ifdim\lastskip=0pt \ifdim\lastkern=0pt \ifnum\lastpenalty=0
10.      \global\setbox0=\lastbox \ifvoido
11.        \else \Anylefttrue
12.          \ifhbox0 \global\advance\lineno1 \fi \fi % count \hboxes on the page
13.        \else \Anylefttrue \unpenalty \fi
14.        \else \Anylefttrue \unkern \fi
15.        \else \Anylefttrue \unskip \fi
16.    \ifAnyleft \repeat}
17.
18. \output={\setbox\brk=\vbox{\unvcopy255 \countlines}
19.           \ifdim\ht\brk>0pt \message{Incomplete breakup}
20.             \showboxbreadth=1000 \showbox\brk \fi
21.           \global\setbox\Bsav=\vbox{\vskip\lineno sp \null\unvbox\Bsav}
22.           \plainoutput
23.         }
24.
25. % *** Executable commands ***
26. \zeroToSp
27. \message{Pass 2;} \setbox\Bsav=\vbox{}
28. \vsize=2in % or any desired value
29. \input source \vfill\eject \pageno=1

```

This is a simple pass. It is again divided into declarations and macro definitions (on lines 1-23), and executable commands (on lines 27-29).

Note the \plainoutput on line 22. This causes pages to be shipped out in pass 2, in addition to

the final pages created by pass 3. We end up with two sets of pages that should be identical, except for the footnote numbers. Because of the problem mentioned later, the pages may not be identical, and it is therefore important to compare the two

sets before they are printed. When the results are finally printed, the pages created by pass 2 should, of course, be suppressed.

**Pass 3.** Count variable `\lineshiped` is set to zero. Count variable `\totalines` is set to the first value of `\Bsav` (50 in our example). `\vsize` remains at its normal value. The OTR ships out

```

1. \newcount\totalines \newcount\lineshiped
2.
3. \def\compare{%
4.   \ifnum\lineno>\totalines
5.     \global\lineshiped=\totalines
6.     \global\setbox\Bsav=
7.       \vbox{\unvbox\Bsav \setbox0=\lastbox \global\temp=\lastskip \unskip}%
8.     \global\advance\totalines by \temp
9.     \expandafter\compare % expand recursively for each page w/o footnotes
10.    \fi}
11.
12. \def\Nfootnote#1{%
13.   \setbox\Asav=
14.   \vbox{\unvbox\Asav \setbox0=\lastbox \global\lineno=\lastskip \unskip}%
15.   % extract bottom glue into \lineno
16.   \compare
17.   \advance\lineno -\lineshiped
18.   \footnote{$^{\the\lineno}$}{#1}}
19.
20. \output={\plainoutput}
21.
22. % *** Executable commands ***
23. \message{Pass 3;} \lineshiped=0
24. \setbox\Bsav=
25. \vbox{\unvbox\Bsav \setbox0=\lastbox \global\totalines=\lastskip \unskip}
26. \input source
27. \bye

```

Macro `\compare`, lines 3–10, expands itself recursively to implement the (pseudo-code) loop

```

while \lineno>\totalines
  \lineshiped:=\totalines
  extract \temp from \Bsav
  \totalines:=\totalines+\temp
end while;

```

The `\expandafter` on line 9 makes sure that the `\fi`, on line 10, is gobble up by `TEX` before `\compare` is recursively expanded. Without the `\expandafter`, the `\fi` would be saved in a stack and popped out at the end of the recursion. In case of a deep recursion, that could overflow the stack.

The macros are deliberately kept simple and readable and, as a result, are not completely general, and don't work in all cases. One such case is where

pages in the normal way. Each time `\Nfootnote` is invoked it (1) extracts the next item from `\Asav` into `\lineno`; (2) if  $\lineno \leq \totalines$ , the footnote is created with  $\lineno - \lineshiped$ ; (3) otherwise, `\lineshiped` is set to `\totalines` and the next number is extracted from `\Bsav` and added to `\totalines`. Step (2) is repeated.

there are no footnotes on the first page; there may be other cases. However, in general, this *approach* seems to work, and seems to have just one, small problem. Passes 1 and 2 typeset an asterisk ‘\*’, in the body of the text, where each footnote should be. This is done to occupy space on the line, space that, in pass 3, is taken by the footnote number. Passes 1 and 2 thus end up with the same line breaks *but pass 3 may not*. The problem is that footnote numbers, in our case, are one or two digits, and thus may be slightly wider or narrower than the ‘\*’. This may, in rare cases, cause different line breaks in pass 3, leading to wrong footnote numbers.

**Exercise:** Why is it true that footnote numbers, in our case, can be one or two digits, but not three?

**Saving Numbers in a vbox.** An interesting point is that our line numbers are saved as *glue* in a *\vbox*. This is done by *\vskip\lineno sp \null*. The *sp* is necessary since, otherwise, the value of *\lineno* would be converted to scaled points. The *\null* is an empty *\hbox* to separate the individual pieces of glue in the large *\vbox*. This technique can only be used if the total number of footnotes in the document is not too large. For a large number of footnotes, there may not be enough room in memory for our boxes, and a file should be used (in our case, two files).

The actual saving of the count variable *\lineno* in box *\Asav* is done by:

```
\global\setbox\Asav=
  \vbox{\vskip\lineno sp \null\unvbox\Asav}
```

Extracting the bottom glue item from *\Asav* is done by:

```
\setbox\Asav=
  \vbox{\unvbox\Asav \setbox0=\lastbox
    \global\lineno=\lastskip \unskip}
```

#### Example: Tables Broken Across Pages

Another practical problem\*: In a document with a lot of tables, many times a table is split over two pages. In such a case, the OTR should typeset "Continued..." at the bottom of the page.

Two approaches are shown, one using marks and the other, special boxes, to communicate with the OTR.

The first approach: A *\mark{Continued...}* is inserted at the start of each *\halign* (following the preamble), and a *\mark{}* is inserted just before the end of the table.

The output routine simply typesets *\botmark* at the bottom of the page, using the right font. The following macros are used:

```
\output={\shipout\vbox{\box255
  \smallskip\line{\sevenrm\hfil\botmark}
  \smallskip\line{\the\footline}}
  \advancepageno}
\def\beginCont{\mark{Continued...}}
\def\endCont{\mark{}}
```

and a typical table looks like:

```
\halign{...preamble...\cr \beginCont
...1st line...\cr
...
...last line...\endCont\cr}
```

\* Proposed by Mary McClure.

Note that the first mark becomes part of the first table entry (column 1 row 1). The last mark, similarly, becomes part of the last table entry (last column bottom row). This means that, sometimes, the mark may be locked inside an internal box. For instance, if the preamble says *\$\$*, then the mark will be buried in the math box. Generally this creates no problem but, if the mark is buried too deeply in *\box255*, it may not be discovered [259] during *\shipout*.

A partial remedy is to use *\noalign{\beginCont}* or *\noalign{\endCont}*, depending on which mark is missing during *\shipout*. This way, the mark precedes (or follows) the entire table. The table, in such a case, should end up with ...*<last line>*...*\cr\endCont*. These constructs should be used only in an emergency, since they also may fail. A typical example is a table that starts at the top of a page. Its *\mark{Continued...}* may, in such a case, be the last thing in the preceding page.

An interesting feature of this method is the even page height. Each page shipped out contains a line with the *\botmark*, and this line occupies the same amount of space on the page, regardless of the size of the mark. Thus if the line preceding the mark has a depth of 1.94444pt, and the mark contains the text *Continued...*, (which has a height of 4.78334pt), the *\baselineskip* glue is set at 5.27222pt. This separates the baselines by  $1.94444 + 5.27222 + 4.78334 = 12$ pt. However, if the mark is empty, and the line preceding it has a depth of 0.8333pt, the *\baselineskip* glue right above the mark is set at 11.1667pt, again separating the baselines by  $0.8333 + 11.1667 + 0 = 12$ pt.

#### Communication by Special \vboxes

The second approach uses a *\vbox* with a special depth to communicate with the OTR. This looks promising, especially since the *\vboxes* on both sides of a table can be attached to it by means of a *\nobreak* (which is essentially a *\penalty10000*). The implementation is similar to the preceding case.

```
\def\beginCont{\noalign{\vbox{
  \hrule width0pt height0pt depth1sp}
  \nobreak}}
\def\endCont{\noalign{\nobreak\vbox{
  \hrule width0pt height0pt depth2sp}}}
```

Note that the *\nobreak* in *\beginCont* follows the special *\vbox*, while that in *\endCont* precedes it.

```

\zeroToSp
\newif\ifAnyleft

\def\breakup{%
\loop \Anyleftfalse
\ifdim\lastskip=0pt \ifdim\lastkern=0pt \ifnum\lastpenalty=0
\setbox0=\lastbox \ifvoido
\else \Anylefttrue
\ifvbox0
\ifdim\dp0=1sp \Anyleftfalse \global\toks0={Continued...}
\else\ifdim\dp0=2sp \Anyleftfalse \global\toks0={} \fi \fi
\fi \fi
\else\Anylefttrue \unpenalty \fi
\else \Anylefttrue \unkern \fi
\else \Anylefttrue \unskip \fi
\ifAnyleft \repeat}

\newbox\brk
\output={\setbox\brk=\vbox{\unvcopy255 \breakup}
\ifdim\ht\brk>0pt \message{Incomplete breakup} \fi
\shipout\vbox{\box255\smallskip
\line{\sevenrm\hfil\the\toks0}
\smallskip\line{\the\footline}}
\advancepageno
}


```

This works! Note, however, that macro `\breakup` stops when it finds the first special `\vbox`. In such a case, there is no point in finishing the break up of `\box255`. This method therefore generates many “Incomplete breakup” messages, and the user should make sure that the text and the tables should not contain any of the things that normally stop the breakup.

**Exercise:** A variation of the same problem. Each table is preceded by a header. If the table is broken across pages, the header should be typeset at the top of the second page.

**Exercise:** Add a parameter to macro `\begin-Cont` above. The macro should now create a `\vbox` whose depth is the value of the parameter, in scaled points. Modify macro `\breakup` such that it will save different messages in `\toks0` depending on the depth of the special boxes found.

#### Example: Verse Numbers in the Left Margin

The problem\*: In the Bible, each chapter is divided into verses. If a verse starts on a certain line, we want the verse number typeset on the left margin of the line. Also, if two or more verses start on the

same line, a range of verse numbers, such as 23-24 should be typeset on the left margin.

Solution: Each verse starts with an expansion of macro `\verse`. The macro computes the verse number and typesets it in the body of the text. In addition, it uses a `\vadjust` to generate a special `\vbox` and to attach it, with a `\penalty10000`, right below the line of text in `\box255`. The special box has a height and width of zero, and a depth equal to the verse number in scaled points. A line of text can thus be followed by any number of such boxes, and no page break can occur in that area. The verse numbers are stored in the `\count` variables `\fVerse` (final verse) and `\sVerse` (start verse).

The OTR expands macro `\breakup`, which breaks up `\box255` and transfers its components to `\box1`. On identifying a special `\vbox`, macro `\breakup` expands `\verseline` which (1) converts the depth of the special box into a count; (2) checks for another special box and converts its depth into another count; (3) removes the line of text above the special boxes, attaches the verse number(s) (via `\Label`) as an `\llap`, and adds the result to `\box1`.

After the breakup is complete, the OTR ships out `\box1`.

---

\* Proposed by Robert Batzinger.

```

\newcount\sVerse \newcount\fVerse \newif\iftwo
\def\verseline{%
  \fVerse=\dp0 \unpenalty
  \global\setbox0=\lastbox
  \ifvoid0 \message{error1;}\fi
  \towfalse
  \if vbox0 \ifdim\dp0<500sp \ifdim\dp0>0sp \twottrue \fi \fi \fi
  \iftwo
    \sVerse=\dp0 \unpenalty
    \def\Label{\hbox to .4in{\hfil\the\sVerse--\the\fVerse\hfil}}
    \global\setbox0=\lastbox
    \ifvoid0 \message{error2;}\fi
    \else\def\Label{\hbox to .4in{\hfil\the\fVerse\hfil}}\fi
    \global\setbox1=\vbox{\line{\llap{\sevrm\Label\kern6pt}\box0}\unvbox1}
  }

\newif\ifAnyleft \newcount\pen \newif\ifverseBox

\def\breakup{%
  \loop \Anyleftfalse
    \ifdim\lastskip=0pt \ifdim\lastkern=0pt \ifnum\lastpenalty=0
      \global\setbox0=\lastbox \ifvoid0 % end of breakup loop
    \else \Anylefttrue \verseBoxfalse
      \if vbox0 \ifdim\dp0<500sp \ifdim\dp0>0sp \verseline \verseBoxtrue \fi\fi\fi
      \ifverseBox \else \global\setbox1=\vbox{\box0\unvbox1}\fi \fi
    \else \Anylefttrue \pen=\lastpenalty
      \global\setbox1=\vbox{\penalty\pen\unvbox1} \unpenalty \fi
    \else \Anylefttrue \dimen0=\lastkern
      \global\setbox1=\vbox{\kern\dimen0\unvbox1} \unkern \fi
    \else \Anylefttrue \skip0=\lastskip
      \global\setbox1=\vbox{\vskip\skip0\unvbox1} \unskip \fi
  \ifAnyleft \repeat}

\newbox\brk
\output={\setbox\brk=\vbox{\unvbox255 \breakup}
  \ifdim\ht\brk>0pt \message{Incomplete breakup} \fi
  \setbox1=\vbox to\vsizet{\unvbox1}
  \shipout\box1 \advancepageno}

\newcount\versno \versno=0
\def\verse{%
  \advance\versno by 1
  \hskipitem{\bf\the\versno: }\nobreak
  \vadjust{\nobreak\vbox{\hrule width0pt height0pt depth\the\versno sp}}}

\zeroToSp
\input source
\bye

```

### Problems With This Approach

1. To keep our macros simple, they are limited to at most two verses per line. However, it is easy to generalize `\verseline` to handle up to 3 verses. It is also possible, although probably not necessary, to generalize it to handle any number of verses per line.

**Exercise:** Disregarding the statement above, generalize `\verseline` to handle any number of verses per line. This requires recursive calls to identify and remove any number of consecutive special `\vboxes` below a text line.

2. The verse numbers are typeset on the left margin, centered in an `\hbox` to .4in. This is wide enough for 3-digit verse numbers. For larger numbers, it may be necessary to enlarge that box. If no centering is required, then it is enough to say `\def\Label{\the\sVerse--\the\fVerse}`.

3. The verse numbers always start from 1. It is possible to let the user specify a start number by:

```
\message{enter start verse number:}
\read16to\ent
\versno=\ent
```

instead of `\versno=0`.

4. The macros recognize a special box if its depth is positive and is less than 500sp. In case of many verses, the 500 should be changed to a larger value. The following quote (from [400]) is reassuring: "A distance of 1000sp is invisible to the naked eye."

### Example: Verse Numbers, An Alternative Method

Here is an alternative method that does not transfer components from `\box255` to `\box1`. It breaks up a copy of `\box255` and, each time a special box (or several consecutive special boxes) is discovered, the macro measures the height of the remaining copy, and uses the height to build, in `\box1`, the range of verse numbers in the margin. When the breakup is completed, `\box1` looks like a skeleton with just the verse number ranges. The OTR then superimposes the two boxes by: `\shipout\hbox{\llap{\box1}\box255}`. (A similar method is used on [391–392].)

Here are the steps in detail:

```
\newcount\versno \versno=0
\def\verse{\advance\versno by 1
  \hskip1emf\bf\the\versno: }\nobreak
\vadjust{\nobreak\vbox{
  \hrule width0pt height0pt
  depth\the\versno sp}}}
```

Macro `\verse` creates a special `\vbox` whose depth equals the verse number (in scaled points), and attaches it, with a `\nobreak`, below the line where the verse starts.

The output routine copies `\box255` into `\box2` and expands `\Obreak` to break up `\box2` and create the necessary information in `\box1`. It then invokes `\reversebox` to break up `\box1` and build, again in `\box2`, the correct skeleton. Both steps are described below. The final step is to shipout a superposition of `\box2` and `\box255`.

```
\output={\setbox2=\copy255 \Obreak
\ifdim\ht\brk>0pt
  \message{Incomplete breakup}\fi
\reversebox \setbox2=
\vbox to\vsiz{\unvbox2\vfil}
\wd2=0pt
\shipout\hbox{\llap{\box2}\box255}
\advancepageno}
```

Macro `\Obreak` expands `\breakup` to break up `\box2` and, if another verse (another special box) is found, the height of the remaining `\box2` is placed, as `\kern`, in `\box1`, and `\Obreak` expands itself recursively. The process repeats until no more verses are found on the page.

```
\newbox\brk
\def\Obreak{%
\setbox\brk=\vbox{\unvbox2 \breakup}
\ifanotherverse
  \global\anotherversefalse
  \global\setbox1=
  \vbox{\unvbox1\kern\ht\brk}
  \setbox2=\box\brk
  \expandafter\Obreak
\fi}
```

Macro `\breakup` loops and breaks up items from `\box2` until it reaches the end, or until it finds an item that is a `\vbox` with a depth in the range 0–500sp. If it finds such an item, it expands `\verseline`.

```
\newif\ifAnyleft
\def\breakup{%
\loop \Anyleftfalse
\ifdim\lastskip=0pt \ifdim\lastkern=0pt \ifnum\lastpenalty=0
\global\setbox0=\lastbox \ifvoid0 % end of breakup loop
\else \Anylefttrue
\ifvbox0 \ifdim\dp0<500sp\ifdim\dp0>0sp \verseline \Anyleftfalse \fi\fi\fi
\else \Anylefttrue \unpenalty \fi
\else \Anylefttrue \unkern \fi
\else \Anylefttrue \unskip \fi
\ifAnyleft \repeat}
```

Macro `\verseline` removes the `\penalty10000` that precedes the special box, and checks to see if there is another special box right above it. If there is one, the box and the penalty above it are removed, and the boolean variable `\iftwo` is set to true.

Next, macro `\Label` is defined, as an `\hbox` to `.4in{\hfil one or two verse numbers \hfil}` and is inserted, as an `\llap`, into the margin of `\box1`.

---

```
\newcount\sVerse \newcount\fVerse \newif\iftwo \newif\ifanotherverse

\def\verseline{%
\fVerse=\dp0 \unpenalty
\global\setbox0=\lastbox
\ifvoid0 \message{error1;}\fi
\twofalse
\ifvbox0 \ifdim\dp0<500sp \ifdim\dp0>0sp \twotru \fi \fi \fi
\iftwo
\sVerse=\dp0 \unpenalty
\def\Label{\hbox to .4in{\hfil\the\sVerse--\the\fVerse\hfil}}
\global\setbox0=\lastbox
\ifvoid0 \message{error2;}\fi
\else \def\Label{\hbox to .4in{\hfil\the\fVerse\hfil}} \fi
\global\setbox1=\vbox{\unvbox1 \line{\llap{\severrm\Label\kern6pt}\hfil}}
\global\anotherversetrue}
```

At the end of the process, the output routine expands `\reversebox` to break up items from `\box1`, process them and place them in `\box2` in the correct order. To understand this process, let's imagine a page with three verses at a distance of 2in, 3in and 6in from the top (Figure 2). The breakup process starts at the bottom of the page, measures the height A of verse 3, then B and, finally, C, creating `\box1` as in Figure 3.

However, we want a box that looks like Figures 4-5, where the `\kerns` are measured from one verse to the next, not always from the top. We also have to make sure that the lines of text do not take

any vertical space, so we add a negative `\kern` after each line, to skip back to the top of the line.

```
\def\reversebox{\setbox2=\vbox{%
\ifvoid1
\else
\dimen1=0pt \unvbox1
\loop
\dimen0=\lastkern \unkern
\dimen2=\dimen0
\advance\dimen0 by-\dimen1
\dimen1=\dimen2
\setbox0=\lastbox
\dimen2=\ht0 \advance\dimen2 by\dp0
\repeat}}
```

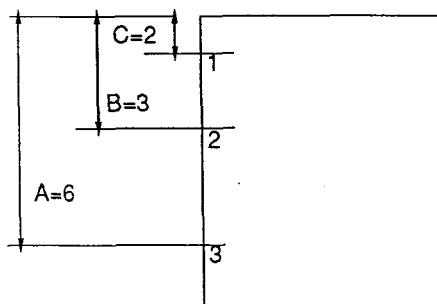


Figure 2

```
\kern 2in (=C)
\hbox{\hfil 3 \hfil}
\kern 3in (=B)
\hbox{\hfil 2 \hfil}
\kern 6in (=A)
\hbox{\hfil 1 \hfil}
```

Figure 3

```
\global\setbox2=\vbox{\unvbox2
\kern\dimen0 \box0 \kern-\dimen2}
\ifdim\lastkern>0pt\repeat
\fi}
```

Macro `\reversebox` contains a loop that breaks up `\box1`, calculates the quantities `C`, `B-C`, `A-B`, and places them in `\box2` with the lines of text, each followed by a negative `\kern`. When finished, The OTR appends a `\vfil` to end up with a height of `\vsiz`. This way, `\box2` has the same height as `\box255`, and they can be superimposed and shipped out together.

To run the whole thing, just say:

```
\zeroToSp
\anotherversefalse
\input source
\bye
```

This is, perhaps, not the most elegant solution, nor is it compact. Each macro, however, has its own, well defined, task, making it easier to read and understand the whole thing.

#### A ‘Special Box’ OTR

The problem: In many modern science texts, the main flow of text is interrupted by ‘special boxes’. They can be used to develop certain topics in detail, to present a historical background of other topics, or to present the author’s opinion or reminiscences.

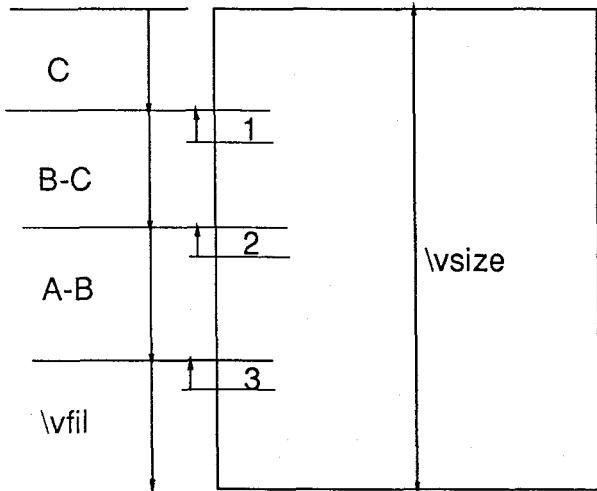


Figure 4

```
\kern 2in (=C)
\hbox{\hfil 1 \hfil}
\kern-(size of preceding line)
\kern 1in (=B-C)
\hbox{\hfil 2 \hfil}
\kern-(size of preceding line)
\kern 3in (=A-B)
\hbox{\hfil 3 \hfil}
\kern-(size of preceding line)
\vfil
```

Figure 5

To distinguish such a box from the rest of the text, it may be surrounded by rules on all sides.

The intuitive approach is to place the special text in a `\vbox` and build the rules as in [Ex. 21.3]. This, of course, won’t work since the ‘special box’ may have to straddle two pages, but a `\vbox` is indivisible.

The approach used here identifies the start and end of the special text by making its lines narrower. Macro `\startspbox` draws the top `\hrule` of the special box and expands `\narrower`. Macro `\endspbox` terminates the effect of `\narrower`, and draws the bottom `\hrule`. Note that the hrules are placed in boxes, since otherwise they would cause an incomplete breakup.

The OTR breaks up the page and creates a duplicate. Each narrow line (a line for which `\rightskip > 0`) is surrounded with two short rules. To make the rules on successive lines touch, the normal interline glue is suppressed when a narrow line is found.

```

\def\Hrule{\line{\vrule width\hsize height.4pt}}
\def\startspbox{\medskip \Hrule \nobreak \smallskip \begingroup \narrower}
\def\endspbox{\smallskip \nobreak \endgroup \Hrule \medskip}

\zeroToSp

\newif\ifsurround
\def\Strut{\vrule height8.5pt depth3.5pt}
\def\checkline{%
  \setbox2=\hbox{\unhcopy0
    \ifdim\lastskip>0pt \global\surroundtrue
    \else\global\surroundfalse\fi}%

\newif\ifAnyleft \newcount\pen
\def\specialbox{%
  \loop \Anyleftfalse
    \ifdim\lastskip=0pt \ifdim\lastkern=0pt \ifnum\lastpenalty=0
      \global\setbox0=\lastbox \ifvoid0 % end of breakup loop
    \else \Anylefttrue
      \ifhbox0\checkline \ifsurround\setbox0=\hbox{\Strut\box0\Strut}\fi \fi
      \global\setbox1=\vbox{\box0 \unvbox1} \fi
    \else \Anylefttrue \pen=\lastpenalty
      \global\setbox1=\vbox{\penalty\pen\unvbox1} \unpenalty \fi
    \else \Anylefttrue \dimen0=\lastkern
      \global\setbox1=\vbox{\kern\dimen0 \unvbox1} \unkern \fi
    \else \Anylefttrue \skip0=\lastskip
      \ifsurround\skip0=0pt \fi % suppress the normal interline glue
      \global\setbox1=\vbox{\vskip\skip0 \unvbox1} \unskip \fi
    \fi
  \ifAnyleft \repeat

\newbox\brk
\output={\setbox\brk=\vbox{\unvbox255 \specialbox}
  \ifdim\ht\brk>0pt \message{Incomplete breakup} \fi
  \shipout\box1 \advancepageno}

```

### Example: Revision Bars

Certain documents—such as the bylaws of an organization, or the user's manual for a computer system—may go through many revisions. Sometimes it is desirable to emphasize (or flag) the revised parts by placing a vertical bar on the left margin of revised lines. If the revision is short, affecting only one line, there is no need for a special OTR and a \vadjust like the one below, can be used (see also [Ex. 14.28]).

```
\def\rev{\vadjust{\moveleft6pt\vbox to0pt{
  \kern-12pt\hrule height10pt width1pt\vss}}}
```

However, if the revision may affect more than one line, the problem becomes much more complex and the OTR should be involved.

**A Simple Method.** We start with a relatively simple approach,\* which is sketched below, but is not implemented.

1. Macro \beginvbars saves the page-so-far in a box \partialpage.
2. Macro \endvbars places a bar on the left of \box255, appends it to \partialpage, and returns the whole thing to the MVL, so that a good page break can be found.

The problems with this approach are:

1. The revision may start in mid-paragraph. In such a case, the first part of the paragraph goes in box \partialpage, and eventually has to be seamlessly glued to the rest of the paragraph. A similar case occurs when the revision ends within a paragraph.

---

\* Due to Amy Hendrickson.

2. When `\box255` is appended to `\partial-page`, its `\topskip` glue should be replaced by the normal interline skip.

**A Better Solution.** The approach shown here is different. The start and end of each revision are flagged with small, special boxes placed between the text lines. The OTR breaks up `\box255` looking for the special boxes. The distance of each special box from the top of the page is measured. The distances are then used to prepare vertical rules in a separate box (`\box3`), which is eventually typeset on the left of `\box255`.

Macro `\startrev` uses `\vadjust` to place a special `\vbox` with a height of `1sp` below the line where the revision starts. Macro `\endrev` places a similar box, with a height of `2sp`, below the last line of the revised text. Note that, if the revised text

is short, the two special boxes may end up being placed, one above the other, below the same line.

```
\def\startrev{\vadjust{%
\nointerlineskip\nobreak\vbox to1sp{}}
\def\endrev{\vadjust{%
\nointerlineskip\nobreak\vbox to2sp{}}}}
```

Macro `\Obreak` expands `\breakup` with a copy of `\box255`. The breakup loop stops when a special box, with height = `2sp` is found. `\Obreak` then measures the height of the remaining page, stores that height in `\box1` as `\kern`, stores a flag indicating that a vertical bar should end at that point, and restarts the loop. When a box with height = `1sp` is found, `\Obreak` does a similar thing, except that it places a different flag, indicating that the bar should start at that point. The flags are special hboxes with a width of either `1sp` or `2sp`.

```
\zeroToSp

\output={\global\setbox1=\vbox{}
\setbox2=\copy255 \Obreak
\ifdim\ht\brk>0pt \message{Incomplete breakup} \fi
\arrangebox
\setbox3=\vbox to\vsized{\unvbox3\vfill} \wd3=6pt
\shipout\hbox{\llap{\box3}\box255}
\advancepageno}

\newbox\brk
\newif\ifstartbar \startbarfalse \newif\ifendbar \endbarfalse
\def\Obreak{%
\setbox\brk=\vbox{\unvbox2 \breakup}
\ifstartbar
\global\startbarfalse
\global\setbox1=\vbox{\unvbox1\kern\ht\brk\hbox to1sp{}}
\setbox2=\box\brk
\expandafter\Obreak
\fi
\ifendbar
\global\endbarfalse
\global\setbox1=\vbox{\unvbox1\kern\ht\brk\hbox to2sp{}}
\setbox2=\box\brk
\expandafter\Obreak
\fi
}

\newif\ifAnyleft
\def\breakup{%
\loop \Anyleftfalse
\ifdim\lastskip=0pt \ifdim\lastkern=0pt \ifnum\lastpenalty=0
\global\setbox0=\lastbox \ifvoido
\else \Anylefttrue
```

```

\ifvbox0
  \ifdim\ht0=1sp \global\startbartrue \Anyleftfalse \fi
  \ifdim\ht0=2sp \global\endbartrue \Anyleftfalse \fi
\fi \fi
\else \Anylefttrue \unpenalty \fi
\else \Anylefttrue \unkern \fi
\else \Anylefttrue \unskip \fi
\ifAnyleft \repeat}

```

At the end of the breakup loop, the OTR expands macro `\arrangebox`, which reads the kerns and flags from `\box1`, and uses them to generate the actual vertical bars in `\box3`. It uses the following algorithm:

```

PrevKern:=0;
read Kern,Flag from \box1
if Flag=start
  place \kern of size Kern-PrevKern
  in \box3
PrevKern:=Kern, PrevFlag:=Flag;
if Flag=end
  place a rule of size Kern-PrevKern
  in \box3
PrevKern:=Kern, PrevFlag:=Flag;
if \box1 is empty and PrevFlag=start
  place a rule of size \vsize-PrevKern
  in \box3,
End;

```

And here is a listing:

```

\newif\ifcontin
\def\arrangebox{
  \setbox3=\vbox{} \dimen1=0pt
  \loop
    \ifdim\ht1>0pt
      \setbox1=\vbox{\unvbox1
        \global\setbox0=\lastbox
        \global\skip0=\lastkern \unkern}
      \contintrue \dimen0=\wd0
      \dimen2=\skip0
      \advance\dimen2 by-\dimen1
      \dimen1=\skip0
      \ifdim\dimen0=1sp
        \setbox3=\vbox{\unvbox3 \kern\dimen2}
      \fi
      \ifdim\dimen0=2sp
        \setbox3=\vbox{\unvbox3
          \hrule height\dimen2 width1pt}
      \fi
    \else
      \continfalse
      \ifdim\dimen0=1sp \dimen2=\vsize
        \advance\dimen2 by-\dimen1
      \fi
    \fi
  \repeat}

```

```

\setbox3=\vbox{\unvbox3
  \hrule height\dimen2 width1pt}
\fi
\fi
\ifcontin \repeat}

```

As usual, the macros can be improved. The user may notice that the size and placement of the bars is not ideal, and can be improved. This is especially true for cases where only one line of text is revised.

**Exercise:** Generalize the macros so that they can typeset a revision number, in `\sevenrm`, on the left of each bar. The number should be specified by the user, as a parameter of `\startrev`.

## Summary

The examples and techniques described here, even though incomplete and simplified, demonstrate how very powerful `TEX` is, compared to other typesetting systems.

The main concepts behind `TEX` namely, boxes, glue, penalties and macros, are different from those used by other systems, and are more difficult to master. At the same time, they are more powerful, and the user who is willing to invest the time and effort necessary to learn `TEX`, is rewarded by high quality results.

Part III will introduce insertions and their use in OTRs. There will be a general introduction to insertions, examples of OTRs with insertions, and a description of the plain format OTR.

◊ David Salomon  
 California State University,  
 Northridge  
 Computer Science Department  
 Northridge, CA 91330  
`dxs@mx.csun.edu`

## Macros

### Lists in *TeX*'s Mouth

Alan Jeffrey

#### 1 Why lists?

Originally, I wanted lists in *TeX* for a paper I was writing which contained a lot of facts.

**Fact i** *Cows have four legs.*

**Fact ii** *People have two legs.*

**Fact iii** *Lots of facts in a row can be dull.*

These are generated with commands like

```
\begin{fact}
```

```
\Forward{Fac-yawn}
```

Lots of facts in a row can be dull.

```
\end{fact}
```

I can then refer to these facts by saying

```
\By[Fac-yawn, Fac-cows, Fac-people]
```

to get [i, ii, iii]. And as if by magic, the facts come out sorted, rather than in the jumbled order I typed them. This is very useful, as I can reorganize my document to my heart's content, and not have to worry about getting my facts straight.

Originally I tried programming this sorting routine in *TeX*'s list macros, from Appendix D of *The TeXbook*, but I soon ran into trouble. The problem is that all the Appendix D macros work by assigning values to macros. For example:

```
\concatenate\foo=\bar\baz
```

expands out to

```
\ta=\expandafter{\bar}
```

```
\tb=\expandafter{\baz}
```

```
\edef\foo{\the\ta\the\tb}
```

which assigns the macro `\foo` the contents of `\bar` followed by the contents of `\baz`. Programming sorting routines (which are usually recursive) in terms of these lists became rather painful, as I was constantly having to watch out for local variables, worrying about what happened if a local variable had the same name as a global one, and generally having a hard time.

Then I had one of those "flash of light" experiences — "You can do lambda-calculus in *TeX*," I thought, and since you can do lists directly in lambda calculus, you should be able to do lists straightforwardly in *TeX*. And so you can. Well, fairly straightforwardly anyway.

So I went and did a bit of mathematics, and derived the *TeX* macros you see here. They were for-

mally verified, and worked first time (modulo typing errors, of which there were two).

#### 2 *TeX*'s mouth and *TeX*'s stomach

*TeX*'s programming facilities come in two forms — there are *TeX*'s *macros* which are expanded in its mouth, and some additional *assignment* operations like `\def` which take place in the stomach. *TeX* can often spring surprises on you as exactly what gets evaluated where. For example, in *LATEX* I can put down a label by saying `\label{Here}`. Then I can refer back to that label by saying `Section^{\ref{Here}}`, which produces Section 2. Unfortunately, `\ref{Here}` does *not* expand out to 2! Instead, it expands out to:

```
\edef\@tempa{\@nameuse{r@Here}}
\expandafter\@car\@tempa@nil\null
```

This means that I can't say

```
\ifnum\ref{Here}<4 Hello\fi
```

and hope that this will expand out to Hello. Instead I get an error message. Which is rather a pity, as *TeX*'s mouth is quite a powerful programming language (as powerful as a Turing Machine in fact).

#### 3 Functions

A *function* is a mathematical object that takes in an argument (which could well be another function) and returns some other mathematical object. For example the function *Not* takes in a boolean and returns its complement. I'll write function application without brackets, so *Not b* is the boolean complement of *b*.

Function application binds to the left, so *f a b* is (*f a*) *b* rather than *f(a b)*. For example, *Or a b* is the boolean or of *a* and *b*, and *Or True* is a perfectly good function that takes in a boolean and returns *True*.

The obvious equivalents of functions in *TeX* are macros — if I define a function *Foo* to be:

```
Foo x = True
```

then it can be translated into *TeX* as:

```
\def\Foo#1{\True}
```

So where *Foo* is a function that takes in one argument, `\Foo` is a macro that takes in one parameter. Nothing has changed except the jargon and the font. *TeX* macros can even be partially applied, for example if we defined:

```
Baz = Or True
```

then the *TeX* equivalent would be

```
\def\Baz{\Or\True}
```

Once `\Baz` is expanded, it will expect to be given a parameter, but when we are defining things, we can go around partially applying them all we like.

Here, I'm using `=` without formally defining it, which is rather naughty. If I say  $x = y$ , this means "given enough parameters,  $x$  and  $y$  will eventually expand out to the same thing." For example  $\text{Foo} = \text{Baz}$ , because for any  $x$ ,

$$\begin{aligned}\text{Foo } x \\ &= \text{True} \\ &= \text{Or True } x \\ &= \text{Baz } x\end{aligned}$$

Normally, functions have to respect equality which means that:

- if  $x = y$  then  $\text{f } x = \text{f } y$ , and
- if  $x$  respects equality, then  $\text{f } x$  respects equality.

However, some `\TeX` control sequences don't obey this. For example, `\string\Foo` and `\string\Baz` are different, even though  $\text{Foo} = \text{Baz}$ . Hence `string` doesn't respect equality. Unless otherwise stated, we won't assume functions respect equality, although all the functions defined here do.

All of our functions have capital letters, so that their `\TeX` equivalents (`\Not`, `\Or` and so on) don't clash with standard `\TeX` or `\LaTeX` macros.

### 3.1 Identity

The simplest function is the *identity* function, called `Identity` funnily enough, which is defined:

$$\text{Identity } x = x$$

This, it must be admitted, is a pretty dull function, but it's a useful basic combinator. It can be implemented in `\TeX` quite simply.

---

```
\def\Identity{\lambda x.x}
```

---

The rules around this definition mean that it is actually part of `Lambda.sty` and not just another example.

### 3.2 Error

Whereas `Identity` does nothing in a fairly pleasant sort of way, `Error` does nothing in a particularly brutal and harsh fashion. Mathematically, `Error` is the function that destroys everything else in front of it. It is often written as  $\perp$ .

$$\text{Error } x = \text{Error}$$

In practice, destroying the entire document when we hit one error is a bit much, so we'll just print out an error message. The user can carry on past an error at their own risk, as the code will no longer be formally verified.

---

```
\def\Error{\lambda x.\errmessage{Abandon verification all ye who enter here}}
```

---

Maybe this function ought to return a more useful error message ...

### 3.3 First and Second

Two other basic functions are *First* and *Second*, both of which take in two arguments, and do the obvious thing. They are defined:

$$\begin{aligned}\text{First } x y &= x \\ \text{Second } x y &= y\end{aligned}$$

We could, in fact, define *Second* in terms of *Identity* and *First*. For any  $x$  and  $y$ ,

$$\begin{aligned}\text{First Identity } x y \\ &= \text{Identity } y \\ &= y \\ &= \text{Second } x y\end{aligned}$$

So  $\text{First Identity} = \text{Second}$ . This means that anywhere in our `\TeX` code we have `\First\Identity` we could replace it by `\Second`. This is perhaps not the most astonishing `\TeX` fact known to humanity, but this sort of proof did enable more complex bits of `\TeX` to be verified before they were run.

The `\TeX` definitions of `\First` and `\Second` are pretty obvious.

---

```
\def\First{\lambda x x}
\def\Second{\lambda x x}
```

---

Note that in `\TeX` `\First\foo\bar` expands out to `\foo` *without* expanding out `\bar`. This is very useful, as we can write macros that would take forever and a day to run if they expanded all their arguments, but which actually terminate quite quickly. This is called *lazy evaluation* by the functional programming community.

### 3.4 Compose

Given two functions  $f$  and  $g$  we would like to be able to *compose* them to produce a function that first applies  $g$  then applies  $f$ . Normally, this is written as  $f \circ g$ , but unfortunately `\TeX` doesn't have infix functions, so we'll have to write it `Compose f g`.

$$\text{Compose } f g x = f(g x)$$

From this definition, we can deduce that `Compose` is associative:

$$\begin{aligned}\text{Compose } (\text{Compose } f g) h \\ &= \text{Compose } f (\text{Compose } g h)\end{aligned}$$

and *Identity* is the left unit of *Compose*:

$$\text{Compose Identity } f = f$$

The reader may wonder why *Identity* is called a *left* unit even though it occurs on the right of the *Compose*—this is a side-effect of using prefix notations where infix is more normal. The infix version of this equation is:

$$\text{Identity} \circ f = f$$

so *Identity* is indeed on the left of the composition.

*Compose* can be implemented in TeX as

---

```
\def\Compose{\#1\#2\#3{\#1{\#2{\#3}}}}
```

---

### 3.5 Twiddle

Yet another useful little function is *Twiddle*, which takes in a function and reverses the order that function takes its (first two) arguments.

$$\text{Twiddle } f x y = f y x$$

Again, there aren't many immediate uses for such a function, but it'll come in handy later on. It satisfies the properties

$$\text{Twiddle First} = \text{Second}$$

$$\text{Twiddle Second} = \text{First}$$

$$\text{Compose Twiddle Twiddle} = \text{Identity}$$

Its TeX equivalent is

---

```
\def\Twiddle{\#1\#2\#3{\#1{\#3}{\#2}}}
```

---

This function is called “twiddle” because it is sometimes written  $\tilde{f}$  (and  $\sim$  is pronounced “twiddle”). It also twiddles its arguments around, which is quite nice if your sense of humour runs to appalling puns.

## 4 Booleans

As we're trying to program a sorting routine, it would be nice to be able to define orderings on things, and to do this we need some representation of boolean variables. Unfortunately TeX doesn't have a type for booleans, so we'll have to invent our own. We'll implement a boolean as a function  $b$  of the form

$$b x y = \begin{cases} x & \text{if } b \text{ is true} \\ y & \text{otherwise} \end{cases}$$

More formally, a boolean  $b$  is a function which respects equality, such that for all  $f$ ,  $g$  and  $z$ :

$$b f g z = b(f z)(g z)$$

and for all  $f$  and  $g$  which respect equality,

$$b(f b)(g b) = b(f \text{First})(g \text{Second})$$

All the functions in this section satisfy these properties. Surprisingly enough, so does *Error*, which is quite useful, as it allows us to reason about booleans which “go wrong”.

### 4.1 True, False and Not

Since we are implementing booleans as functions, we already have the definitions of *True*, *False* and *Not*.

$$\text{True} = \text{First}$$

$$\text{False} = \text{Second}$$

$$\text{Not} = \text{Twiddle}$$

So for free we get the following results:

$$\text{Not True} = \text{False}$$

$$\text{Not False} = \text{True}$$

$$\text{Compose Not Not} = \text{Identity}$$

The TeX implementation is not exactly difficult:

---

```
\let\True=\First
\let\False=\Second
\let\Not=\Twiddle
```

---

### 4.2 And and Or

The definitions of *And* and *Or* are:

$$\text{And } a b = \begin{cases} b & \text{if } a \text{ is true} \\ \text{False} & \text{otherwise} \end{cases}$$

$$\text{Or } a b = \begin{cases} \text{True} & \text{if } a \text{ is true} \\ b & \text{otherwise} \end{cases}$$

With our definition of what a boolean is, this is just the same as

$$\text{And } a b = a b \text{ False}$$

$$\text{Or } a b = a \text{ True } b$$

From these conditions, we can show that *And* is associative, and has left unit *True* and left zeros *False* and *Error*:

$$\text{And } (\text{And } a b) c = \text{And } a (\text{And } b c)$$

$$\text{And True } b = b$$

$$\text{And False } b = \text{False}$$

$$\text{And Error } b = \text{Error}$$

*Or* is associative, has left unit *False* and left zeros *True* and *Error*:

$$\text{Or } (\text{Or } a b) c = \text{Or } a (\text{Or } b c)$$

$$\text{Or False } b = b$$

$$\text{Or True } b = \text{True}$$

$$\text{Or Error } b = \text{Error}$$

De Morgan's laws hold:

$$\text{Not } (\text{And } a b) = \text{Or } (\text{Not } a) (\text{Not } b)$$

$$\text{Not } (\text{Or } a b) = \text{And } (\text{Not } a) (\text{Not } b)$$

and *And* and *Or* left-distribute through one another:

$$\text{Or } a (\text{And } b c) = \text{And } (\text{Or } a b) (\text{Or } a c)$$

$$\text{And } a (\text{Or } b c) = \text{Or } (\text{And } a b) (\text{And } a c)$$

*And* and *Or* are *not* commutative, though. For example,

$$\begin{aligned} \text{Or True Error} \\ = & \quad \text{True True Error} \\ = & \quad \text{True} \end{aligned}$$

but

$$\begin{aligned} \text{Or Error True} \\ = & \quad \text{Error True True} \\ = & \quad \text{Error} \end{aligned}$$

This is actually quite useful since there are some booleans that need to return an error occasionally. If *a* is *True* when *b* is safe (i.e. doesn't become *Error*) and is *False* otherwise, we can say *Or a b* and know we're not going to get an error. This is handy for things like checking for division by zero, or trying to get the first element of an empty list.

Similarly, because of the possibility of *Error*, *And* and *Or* don't right-distribute through each other, as

$$\begin{aligned} \text{Or (And False Error) True} \\ \neq \quad \text{And (Or False True) (Or Error True)} \end{aligned}$$

As errors shouldn't crop up, this needn't worry us too much.

---

```
\def\And#1#2{\#1{\#2}\False}
\def\Or#1#2{\#1\True{\#2}}
```

---

### 4.3 Lift

Quite a lot of the time we won't be dealing with booleans, but with *predicates*, which are just functions that return a boolean. For example, the predicate *Lessthan* is defined below so that *Lessthan i j* is true whenever *i < j*. Given a predicate *p* we would like to be able to *lift* it to *Lift p*, defined:

$$\text{Lift } p f g x = p x f g x$$

For example, *Lift (Lessthan 0) f g* takes in a number and applies *f* to it if it is positive and *g* to it otherwise. This is quite useful for defining functions.

---

```
\def\Lift#1#2#3#4{\#1{\#4}{\#2}{\#3}{\#4}}
```

---

### 4.4 Lessthan and TeXif

Finally, we would like to be able to use *TeX*'s built-in booleans as well as our own. For example, we would like a predicate *Lessthan* such that:

$$\text{Lessthan } i j = \begin{cases} \text{True} & \text{if } i < j \\ \text{False} & \text{if } i \geq j \\ \text{Error} & \text{otherwise} \end{cases}$$

The *Error* condition happens if we try applying *Lessthan* to something that isn't a number —

*Lessthan True False* is *Error*.<sup>1</sup> This is fine as a mathematical definition, but how will we implement it? If we assume we have a macro *\TeXif*, which converts *TeX* if-statements into booleans, we could just define:

---

```
\def\Lessthan#1#2{\TeXif{\ifnum#1<#2 }}
```

---

So the question is just how to define *\TeXif*. Unfortunately, the "obvious" code does not work:

```
\def\TeXif#1#2#3{\#1#2\else#3\fi}
```

For example, *\TeXif\iftrue\True\True* doesn't expand out to *\True*. Instead, it expands as:

```
\TeXif\iftrue\True\True
= \iftrue\True\else\True\fi
= \True\else\True\fi
= \else\fi
=
```

Another common *TeX*nique is to use a macro *\next* to be the expansion text:

```
\def\TeXif#1#2#3%
{\#1\def\next{\#2}\else\def\next{\#3}\fi
 \next}
```

However, this uses *TeX*'s stomach to do the *\def*, and we are trying to do this using only the mouth. One (slightly tricky) solution is to use pattern-matching to gobble up the offending *\else* and/or *\fi*.

---

```
\def\gobblefalse\else\gobbletrue\fi#1#2%
{\fi#1}
\def\gobbletrue\fi#1#2%
{\fi#2}
\def\TeXif#1%
{\#1\gobblefalse\else\gobbletrue\fi}
```

---

So if the *TeX* if-statement is true, *\gobblefalse* gobbles up the false-text, otherwise *\gobbletrue* gobbles up the true-text. For example,

```
\TeXif\iftrue\True\True
= \iftrue\gobblefalse\else
   \gobbletrue\fi\True\True
= \gobblefalse\else
   \gobbletrue\fi\True\True
= \fi\True
= \True
```

Phew. And so we have booleans.

<sup>1</sup> Actually, that's a little white lie — trying to persuade *TeX* to do run-time type checking isn't much fun. So the *TeX* implementation of this is actually a *refinement* where the *Error* condition has been replaced by whatever it is *TeX* does if you try doing *\ifnum x < y* when *x* and *y* aren't numbers.

## 5 Lists

A list is a (possibly infinite) sequence of values. For example, the list [1, 2, 3] contains three numbers, the list [] contains none, and the list [1, 2, 3, ...] contains infinitely many. A list is either *empty* (written []) or is comprised of a *head*  $x$  and a *tail*  $xs$  (in which case it's written  $x : xs$ ). For example,  $1 : 2 : 3 : []$  is [1, 2, 3].

In a similar fashion to the implementation of booleans, a list  $xs$  is implemented as a function of the form

$$xs f e = \begin{cases} e & \text{if } xs \text{ is empty} \\ f y ys & \text{if } xs \text{ has head } y \text{ and tail } ys \end{cases}$$

Again, we are implementing a datatype as a function, a quite powerful trick, just not one usually seen in TeX. We will assume that whenever a list  $x : xs$  is applied to  $f$  and  $e$ ,  $f x$  respects equality. This allows us to assume that if  $xs = ys$  then  $x : xs = x : ys$ , which is handy.

### 5.1 Nil, Cons, Stream and Singleton

The simplest list is *Nil*, the empty list which we have been writing [].

$$\text{Nil} = \text{Second}$$

The other possible list is *Cons*  $x$   $xs$ , which has head  $x$  and tail  $xs$ .

$$\text{Cons } x \text{ } xs \text{ } f \text{ } e = f \text{ } x \text{ } xs$$

Every list can be constructed using these functions. The list [1, 2, 3] is *Cons* 1 (*Cons* 2 (*Cons* 3 *Nil*)), and the list [a, a, a, ...] is *Stream* a where *Stream* is defined:

$$\text{Stream } a = \text{Cons } a (\text{Stream } a)$$

There's even at least one application for infinite lists, as we'll see in Section 7.

The singleton list [a] is *Singleton* a, defined as:

$$\text{Singleton } a = \text{Cons } a \text{ Nil}$$

These all have straightforward TeX definitions.

---

```
\let\Nil=\Second
\def\Cons#1#2#3#4{\#3{\#1}{\#2}}
\def\Stream#1{\{ \Cons{\#1}{\Stream{\#1}} \}}
\def\Singleton#1{\{ \Cons{\#1}{\Nil} \}}
```

---

### 5.2 Head and Tail

So, we can construct any list we like, but we still can't get any information out of it. To begin with, we'd like to be able to get the head and tail of a list.

$$\text{Head } xs = xs \text{ First Error}$$

$$\text{Tail } xs = xs \text{ Second Error}$$

For example, the tail of  $x : xs$  is

$$\begin{aligned} \text{Tail} (\text{Cons } x \text{ } xs) &= \text{Cons } x \text{ } xs \text{ Second Error} \\ &= \text{Second } x \text{ } xs \\ &= xs \end{aligned}$$

The tail of [] is, as one would expect,

$$\begin{aligned} \text{Tail } \text{Nil} &= \text{Nil Second Error} \\ &= \text{Error} \end{aligned}$$

And the head of *Stream* a is

$$\begin{aligned} \text{Head} (\text{Stream } a) &= \text{Stream } a \text{ First Error} \\ &= \text{Cons } a (\text{Stream } a) \text{ First Error} \\ &= \text{First } a (\text{Stream } a) \\ &= a \end{aligned}$$

So we can get the head of an infinite list in finite time. This is fortunate, as otherwise there wouldn't be much point in allowing infinite objects.

---

```
\def\Head#1{\#1\First\Error}
\def\Tail#1{\#1\Second\Error}
```

---

### 5.3 Foldl and Foldr

Using *Head* and *Tail* we can get at the beginning of any non-empty list, but in general we need more information than that. Rather than write a whole bunch of recursive functions on lists, I'll implement two fairly general functions, with which we can implement (almost) everything else.

*Foldl* and *Foldr* both take in functions and apply them recursively to a list. *Foldl* starts at the left of the list, and *Foldr* starts at the right. For example,

$$\begin{aligned} \text{Foldl } f \text{ } e [1, 2, 3] &= f (f (f e 1) 2) 3 \\ \text{Foldr } f \text{ } e [1, 2, 3] &= f 1 (f 2 (f 3 e)) \end{aligned}$$

These functions will be used a lot later on. *Foldl* can be defined:

$$\begin{aligned} \text{Foldl } f \text{ } e \text{ } xs &= xs (\text{Foldl}' \text{ } f \text{ } e) \text{ } e \\ \text{Foldl}' \text{ } f \text{ } e \text{ } xs &= \text{Foldl } f (\text{f } e \text{ } xs) \text{ } xs \end{aligned}$$

So *Foldl*  $f$  [] is

$$\begin{aligned} \text{Foldl } f \text{ } e \text{ Nil} &= \text{Nil} (\text{Foldl}' \text{ } f \text{ } e) \text{ } e \\ &= e \end{aligned}$$

And *Foldl*  $f$   $(x : xs)$  is

$$\begin{aligned} \text{Foldl } f \text{ } e (\text{Cons } x \text{ } xs) &= \text{Cons } x \text{ } xs (\text{Foldl}' \text{ } f \text{ } e) \text{ } e \\ &= \text{Foldl}' \text{ } f \text{ } e \text{ } x \text{ } xs \\ &= \text{Foldl } f (\text{f } e \text{ } x) \text{ } xs \end{aligned}$$

For example,  $\text{Foldl } f \ e [1, 2, 3]$  is

$$\begin{aligned} \text{Foldl } f \ e [1, 2, 3] &= \text{Foldl } f (\text{f } e 1) [2, 3] \\ &= \text{Foldl } f (\text{f } (\text{f } e 1) 2) [3] \\ &= \text{Foldl } f (\text{f } (\text{f } (\text{f } e 1) 2) 3) [] \\ &= \text{f } (\text{f } (\text{f } e 1) 2) 3 \end{aligned}$$

as promised. Similarly, we can define  $\text{Foldr}$  as

$$\begin{aligned} \text{Foldr } f \ e xs &= xs (\text{Foldr}' \ f \ e) e \\ \text{Foldr}' \ f \ e xs &= f \ x (\text{Foldr } f \ e xs) \end{aligned}$$

For  $\text{Foldr } f$  to respect equality,  $f \ x$  should respect equality.

When we do the unfolding, we discover that

$$\begin{aligned} \text{Foldr } f \ e [] &= e \\ \text{Foldr } f \ e (x : xs) &= f \ e (\text{Foldr } f \ e xs) \end{aligned}$$

$\text{Foldr}$  tends to be more efficient than  $\text{Foldl}$ , because  $\text{Foldl}$  has to run along the entire list before it can start applying  $f$ , whereas  $\text{Foldr}$  can apply  $f$  straight away. If  $f$  is a lazy function, this can make quite a difference.  $\text{Foldl}$  on infinite lists, anyone?

---

```
\def\Foldl#1#2#3%
{#3{\Foldl@{#1}{#2}}{#2}}
\def\Foldl@#1#2#3#4%
{ \Foldl@{#1}{#1{#2}{#3}}{#4}}
\def\Foldr#1#2#3%
{#3{\Foldr@{#1}{#2}}{#2}}
\def\Foldr@#1#2#3#4%
{#1{#3}{\Foldr@{#1}{#2}}{#4}}
```

---

#### 5.4 Cat

Given two lists, we would like to be able to stick them together, which is what  $\text{Cat}$  (short for “concatenate”) does. For example,  $\text{Cat} [1, 2] [3, 4]$  is  $[1, 2, 3, 4]$ . It can be defined using  $\text{Foldr}$ :

$$\text{Cat } xs \ ys = \text{Foldr } \text{Cons } ys \ xs$$

So

$$\begin{aligned} \text{Cat} [1, 2] [3, 4] &= \text{Foldr } \text{Cons} [3, 4] [1, 2] \\ &= \text{Cons } 1 (\text{Foldr } \text{Cons} [3, 4] [2]) \\ &= \text{Cons } 1 (\text{Cons } 2 (\text{Foldr } \text{Cons} [3, 4] [])) \\ &= \text{Cons } 1 (\text{Cons } 2 [3, 4]) \\ &= [1, 2, 3, 4] \end{aligned}$$

The  $\text{\TeX}$  code for  $\text{\Cat}$  is suspiciously similar to its mathematical definition.

---

```
\def\Cat#1#2{\Foldr\Cons{#2}{#1}}
```

---

#### 5.5 Reverse

We can reverse any list with the function  $\text{Reverse}$ , defined using  $\text{Foldl}$ :

$$\text{Reverse} = \text{Foldl } (\text{Twiddle } \text{Cons}) \text{ Nil}$$

For example,  $\text{Reverse} [1, 2, 3]$  can be calculated:

$$\text{Reverse} [1, 2, 3]$$

$$\begin{aligned} &= \text{Foldl } (\text{Twiddle } \text{Cons}) \text{ Nil} [1, 2, 3] \\ &= \text{Twiddle } \text{Cons} \\ &\quad (\text{Twiddle } \text{Cons} (\text{Twiddle } \text{Cons} \text{ Nil} 1) 2) 3 \\ &= \text{Cons } 3 (\text{Cons } 2 (\text{Cons } 1 \text{ Nil})) \\ &= [3, 2, 1] \end{aligned}$$

The  $\text{\TeX}$  code for  $\text{\Reverse}$  doesn't even take in any parameters.

---

```
\def\Reverse{\Foldl{\Twiddle\Cons}\Nil}
```

---

#### 5.6 All, Some and Isempy

Given a predicate  $p$ , we can find out if all the elements of a list satisfy  $p$  with  $\text{All } p$ . Similarly we can find if something in the list satisfies  $p$  with  $\text{Some } p$ . For example,

$$\text{All } (\text{Lessthan } 1) [1, 2, 3] = \text{False}$$

$$\text{Some } (\text{Lessthan } 1) [1, 2, 3] = \text{True}$$

These can be defined

$$\text{All } p = \text{Foldr } (\text{Compose And } p) \text{ True}$$

$$\text{Some } p = \text{Foldr } (\text{Compose Or } p) \text{ False}$$

For example,  $\text{Isempy}$  can be defined

$$\text{Isempy} = \text{All } (\text{First False})$$

This is probably not the most efficient check in the world, but we hardly ever need it —  $\text{Foldl}$  or  $\text{Foldr}$  will normally do the job.

---

```
\def\All#1{\Foldr{\Compose\And{#1}}\True}
\def\Some#1{\Foldr{\Compose\Or{#1}}\False}
\def\Isempy{\All{\First\False}}
```

---

#### 5.7 Filter

$\text{Filter}$  takes a predicate  $p$  and a list  $xs$ , and returns a list containing only those elements of  $xs$  that satisfy  $p$ . For example,

$$\text{Filter } (\text{Lessthan } 1) [1, 2, 3] = [2, 3]$$

$\text{Filter}$  can be defined as a  $\text{Foldr}$ :

$$\text{Filter } p = \text{Foldr } (\text{Lift } p \text{ Cons Second}) \text{ Nil}$$

Another easy bit of  $\text{\TeX}$ :

---

```
\def\Filter#1%
{\Foldr{\Lift{#1}\Cons\Second}\Nil}
```

---

## 5.8 Map

*Map* takes a function  $f$  and a list  $xs$  and applies  $f$  to every element of  $xs$ . For example,

$$\text{Map } f [1, 2, 3] = [f 1, f 2, f 3]$$

This is another job for *Foldr*.

$$\text{Map } f = \text{Foldr} (\text{Compose} \text{ Cons } f) \text{ Nil}$$

We shall see *Map* used later on, to convert from a list of names such as [Fac-yawn, Fac-cows], to a list of labels such as [i, iii].

---

```
\def\Map#1{\Foldr{\Compose{\Cons{#1}}}\Nil}
```

---

## 5.9 Insert

The only function we need which isn't easily defined as a reduction is *Insert*, which inserts an element into a sorted list. For example,

$$\text{Insert Lessthan } 3 [1, 2, 4, 5] = [1, 2, 3, 4, 5]$$

*Insert* takes in an ordering as its first parameter, so we're not stuck with one particular order. It is defined directly in terms of the definition of lists.

$$\text{Insert } o \text{ } xs = xs (\text{Insert}' \text{ } o \text{ } x) (\text{Singleton } x)$$

$$\begin{aligned} \text{Insert}' \text{ } o \text{ } x \text{ } ys &= o \text{ } x \text{ } y \\ &\quad (\text{Cons } x (\text{Cons } y \text{ } ys)) \\ &\quad (\text{Cons } y (\text{Insert } o \text{ } x \text{ } ys)) \end{aligned}$$

We can then define the function all this has been leading up to, *Insertsort* which takes an ordering and a list, and insert-sorts the list according to the ordering. For example,

$$\text{Insertsort Lessthan } [2, 3, 1, 2] = [1, 2, 2, 3]$$

We can implement this as a fold:

$$\text{Insertsort } o = \text{Foldr} (\text{Insert } o) \text{ Nil}$$

And so we've got sorted lists.

---

```
\def\Insert#1#2#3%
  {#3{\Insert@{#1}{#2}}{\Singleton{#2}}}
\def\Insert@#1#2#3#4%
  {#1{#2}{#3}%
   {\Cons{#2}{\Cons{#3}{#4}}}%
   {\Cons{#3}{\Insert@{#1}{#2}{#4}}}}
\def\Insertsort#1{\Foldr{\Insert{#1}}\Nil}
```

---

Interestingly, as we have implemented unbounded lists in *TeX*'s mouth, this means we can implement a Turing Machine. So, if you believe the Church-Turing thesis, *TeX*'s mouth is as powerful as any computer anywhere. Isn't that good to know?

## 6 Sorting reference lists

So, these are the macros I've got to play with — how do we apply them to sorting lists of references? Well, I'm using *LATEX*, which keeps the current reference in a macro called *\@currentlabel*, which is 6 at the moment, as this is Section 6. So I just need to store the value of *\@currentlabel* somehow.

Fortunately, I'm only ever going to be making references to facts earlier on in the document, in order to make sure I'm not proving any results in terms of themselves. So I don't need to play around with auxiliary files, and can just do everything in terms of macros.

### 6.1 Number and Label

Each label in the document is given a unique number, in the order the labels were put down. So the number of Fac-cows is *\Number{Fac-cows}*, which expands out to 1, the number of Fac-people is 2, and so on.

Each number has an associated label with it. For example, the first label is *\Label{1}*, which is i, the second label is ii and so on. So to find the label for Fac-cows, we say *\Label{\Number{Fac-cows}}* which expands out to i.

These numbers and labels are kept track of in macros. For example, the number of Fac-cows is kept in *\Number-Fac-cows*. Similarly, the first label is kept in *\Label-1*. As these macros have dashes in their names, they aren't likely to be used already.

So the *TeX* code for *\Number* and *\Label* is pretty simple.

```
\def\Number#1{\csname Number-#1\endcsname}
\def\Label#1{\csname Label-#1\endcsname}
```

### 6.2 Lastnum and Forward

The number of the most recent label is kept in *\Lastnum*.

```
\newcount\Lastnum
```

To put down a label *Foo*, I type *\Forward{Foo}*. This increments the counter *\Lastnum*, and *\xdef* *\Number-Foo* to be the value of *\Lastnum*, which is now 4. So *\Number{Foo}* now expands to 4. Similarly, it *\xdef* *\Label-4* to be *\@currentlabel*, which is currently 6.2. So *\Label{\Number{Foo}}* now expands to 6.2.

```
\def\Forward#1%
  {\global\advance\Lastnum by 1
   \csnameafter\xdef{\Number-#1}%
   {\the\Lastnum}%
   \csnameafter\xdef{\Label-\the\Lastnum}%
   {\@currentlabel}}
```

This uses `\csnameafter\foo\bar`, which expands out to `\foo\bar`.

```
\def\csnameafter#1#2%
  {\expandafter#1\csname#2\endcsname}
```

### 6.3 Listize, Unlistize and Show

At the moment, lists have to be built up using `\Cons` and `\Nil`, which is rather annoying. Similarly, we can't actually do anything with a list once we've built it. We'd like some way of converting lists in the form `[a,b,c]` to and from the form `[a, b, c]`. This is done with `\Listize` and `\Unlistize`. So `\Listize[a,b,c]` expands to

```
\Cons{a}{\Cons{b}{\Cons{c}{\Nil}}}
```

Similarly, `\Unlistize` takes the list `[a, b, c]` and expands out to `[a, b, c]`. `\Unlistize` is done with a `Foldr`.

---

```
\def\Unlistize#1{[#1\Unlistize@{}]}
\def\Unlistize@#1{#1\Foldr\Commaize{}}
\def\Commaize#1#2{, #1#2}
```

---

The macro `\Listize` is just a `TeX` hack with pattern matching. It would have been nice to use `\@ifnextchar` for this, but that uses `\futurelet`, which doesn't expand in the mouth. Oh well.

---

```
\def\Listize[#1]%
  {\Listize@#1,\relax}
\def\Listize@#1,#2]%
  {\TeXif{\ifx\relax#2}%
    {\Singleton{#1}}%
    {\Cons{#1}{\Listize@#2}}}
```

---

This only works for nonempty lists—`\Listize[]` produces the singleton list `\Singleton{}`. It also uses `\relax` as its end-of-list character, so lists with `\relax` in them have to be done by hand. You can't win them all. So

```
$\Unlistize{\Listize[a,b,c]}$
```

produces `[a, b, c]`. This is such a common construction that I've defined a macro `\Show` such that `\Show\foo[a,b,c]` expands out to

```
\Unlistize{\foo{\Listize[a,b,c]}}
```

For example, the equation

$$\text{Filter}(\text{Lessthan } 1)[1, 2, 3] = [2, 3]$$

was generated with

```
\begin{eqnarray*}
 \text{Filter}\,(,\text{Lessthan}\,,1)\,,[1,2,3]
 &=& \text{Show}\,\text{Filter}\{ \text{Lessthan } 1\}[1,2,3]
\end{eqnarray*}
```

Many of the examples in this article were typeset this way.

---

```
\def>Show#1[#2]%
  {\Unlistize{\#1\Listize[#2]}}
```

---

### 6.4 By

Given these macros, we can now sort any list of references with `Bylist`, defined

$$\begin{aligned}
 \text{Bylist } xs &= \text{Map Label} \\
 &\quad (\text{Insertsort Lessthan} \\
 &\quad \quad (\text{Map Number } xs))
 \end{aligned}$$

This takes in a list of label names like `Fac-yawn`, converts it into a list of numbers with `Map Number`, sorts the resulting list with `Insertsort Lessthan`, and finally converts all the numbers into labels like `iii` with `Map Label`. For example,

$$\begin{aligned}
 \text{Bylist } [\text{Fac-yawn}, \text{Fac-cows}] &= \text{Map Label} (\text{Insertsort Lessthan} \\
 &\quad (\text{Map Number } [\text{Fac-yawn}, \text{Fac-cows}])) \\
 &= \text{Map Label} (\text{Insertsort Lessthan } [3, 1]) \\
 &= \text{Map Label } [1, 3] \\
 &= [i, iii]
 \end{aligned}$$

The `TeX` code for this is

```
\def\Bylist#1%
  {\MapLabel
   {\Insertsort\Lessthan
    {\MapNumber{#1}}}}
```

So we can now stick all this together, and define the macro `\By` that prints out lists of references. It is

```
\def\By{\Show\Bylist}
```

So `\By[\text{Fac-yawn}, \text{Fac-cows}]` is `[i, iii]`. Which is quite nice.

## 7 Other applications

Is all this worth it? Well, I've managed to get my lists of facts in order, but that's not the world's most astonishing application. There are other things that these lists are useful for, though.

For example, Damian Cugley has a macro package under development for laying out magazines. MAGTeX's output routine needs to be quite smart, as magazines often have gaps where illustrations or photographs are going to live. In general, each block of text needs to be output in a different fashion from every other block of text. This will be handled by keeping an infinite list of output routines. Each time a box is cut off the scroll to be output, the head of the list is chopped off and is used as the output routine for that box. That way, quite complex page shapes can be built up.

Mainly, though, these macros were written just as a challenge. I learned quite a lot about TeX and needed some TeXniques I'd never seen before. It was also quite pleasing to see that TeX code can be formally verified, albeit in a rather noddy way. Without some sort of abstract view of lists, these TeX macros could not have been written.

## 8 Acknowledgements

Thanks to Jeremy Gibbons for letting me bounce ideas off him and spotting the duff ones, to Damian Cugley for saying "Do you really think TeX is meant to do this?", and to the Problem Solving Club for hearing me out. This work was sponsored by the Science and Engineering Research Council and Hewlett Packard.

◊ Alan Jeffrey  
 Programming Research Group  
 Oxford University  
 11 Keble Road  
 Oxford OX1 3QD  
 Alan.Jeffrey@uk.ac.oxford.prg

## A Nestable Verbatim Mode

Philip Taylor

A few months ago, Sebastian Rahtz asked me if I could make some changes to the verbatim code which he was currently using, and sent me the source. I found it so opaque that I decided to write my own, and the following evolved over a period of a couple of weeks. I would like to acknowledge my debt to Sebastian, and also to Chris Rowley, without whose helpful comments and criticism the code could never have evolved. Of course, the code is now ten times as opaque as that originally used by Sebastian, but at least I understand it (on a good day, when the moon is the seventh house, and Jupiter \haligns with Mars).

The idea is as follows: having said

\input verbatim

at the beginning of one's document, one invokes verbatim mode by

\verbatim <char>

What follows can then contain any character, with the single exception of <char>, and all such text will be copied *verbatim*, with leading spaces retained but invisible, and all embedded spaces retained

and shewn. If <char> is encountered, TeX enters a new inner group (the verbatim environment is itself a group), within which the preceding meaning (i.e. \catcode) of all characters is reinstated. This new inner group continues typesetting in the normal (non-*verbatim*) manner until a further <char> is encountered, whereupon it reverts to verbatim mode; the inner 'normal' mode can itself be interrupted by a further

\verbatim <char>

where <char> can be the same or a different escape character. There is no theoretical limit on the level of nesting, but TeX implementations will invariably run out of space (usually save-stack space) if too many levels are attempted.

To end verbatim mode, one enters inner 'normal' mode through the escape character and then says \mitabrev. Note that this is *not* a reserved string, but simply a macro which expands to {\endgroup \endgroup}; any other name can be chosen if one finds "\mitabrev" unappealing. Thus, at the outermost level, the call and end to \verbatim look like:

\verbatim <char>

<char> \mitabrev

Finally, a mechanism is provided for listing arbitrary files in verbatim mode. If, while in inner 'normal' mode, one says

\AfterGroup {<any balanced text>}

(note the case of \AfterGroup), the <balanced text> will be re-inserted with its original catcodes immediately after the closing <char> which terminates inner 'normal' mode. Thus it will not itself be listed *verbatim*, but will be elaborated according to TeX's normal conventions. Thus if one says

\AfterGroup {\input <filename>}

the contents of the file will be listed in verbatim mode. For example, to list this file itself, one can say

```
\verbatim |
| \AfterGroup {\input verbatim.tex} |
| \mitabrev
```

There remains an anomaly at present: "\ cannot form the escape-character as it will automatically form a <control sequence> with the following character(s) when called with

\verbatim \

I will endeavour to rectify this deficiency in a future release.

The source of `Verbatim.tex` follows.

```

\catcode `@ = 11
\chardef \l@tter = 11
\chardef \other = 12
\newcount \c@unt
\newcount \ch@rcode

\newif \ifd@bugging
\newif \ifshewleadingspaces
\newif \ifshewembeddedspaces
\shewembeddedspacestrue
\ifd@bugging
  \let \m@ssage = \message
\else
  \def \m@ssage #1{%
\fi
%
\def \verbatim #1%
  {\begingroup
    \def \n@sted
      {\begingroup
        \let \n@sted
          = \endgroup
        \onenvironment
        \ignorespaces
      }%
    \tt
    \edef \onenvironment
      {\parindent =
        \the \parindent
        \parskip =
          \the \parskip
        \space
      }%
    \parskip = 0 pt
    \parindent = 0 pt
    \c@unt = 0
    \loop
      \ifnum \c@unt < '\A%
        \s@ve \catcode \c@unt = \other
      \else
        \c@unt = \l@tter
      \fi
      \loop
        \ifnum \c@unt > '\Z%
          \c@unt = \l@tter
        \else
          \c@unt = \other
        \fi
      \repeat
    \endgroup
  }%
}
%
\def \endverbatim {%
  \endgroup
}
%
```

%%% we use commercial-at as a letter throughout;  
 %%% and introduce synonyms for the \catcodes for  
 %%% *letter* and *other*;  
 %%% a loop-counter;  
 %%% this will hold the character-code of the  
 %%% escape character;  
 %%% set *true* if you want to watch the  
 %%% finite-state automaton at work;  
 %%% set *true* if you want to see leading spaces  
 %%% shewn as inverted square cup (explicit);  
 %%% set *false* if you want to see embedded  
 %%% spaces shewn as white space (implicit);  
 %%% if *debugging*,  
 %%% \m@ssage is synonymous with \message  
 %%% otherwise  
 %%% it simply throws its parameter away;

%%% the \verbatim macro takes one parameter  
 %%% and immediately starts a nested group  
 %%% within which \n@sted is defined  
 %%% to start a further group within which  
 %%% \n@sted becomes a synonym for \endgroup

%%% and the environment is restored to that  
 %%% which obtained two levels of nesting out;  
 %%% for tidiness, we ignore any *lwsp*  
 %%% which follows the escape character;  
 %%% we assume Knuth's font-selectors and  
 %%% select the 'typewriter' font;  
 %%% we initialise \onenvironment  
 %%% to prepare to restore \parindent

%%% and \parskip;

%%% and ensure that the value to be assigned to  
 %%% \parskip is properly terminated;  
 %%% we then set \parindent and  
 %%% \parskip to 0 pt;  
 %%% and initialise \c@unt to 0;  
 %%% this loop checks the \catcode of each  
 %%% character code in the range 0...127  
 %%% (or 0...255 for TEX V3) and if it  
 %%% is other than *letter* or *other*, as  
 %%% appropriate, saves the current value in  
 %%% \onenvironment for subsequent restoration  
 %%% within an inner group; it then sets the  
 %%% \catcode to either *letter* or *other*;

```

\else \ifnum \c@unt > '\z%
    \s@ve \catcode \c@unt = \other
\else \ifnum \c@unt > '\Z%
    \ifnum \c@unt < '\a%
        \s@ve \catcode \c@unt = \other
    \else \s@ve \catcode \c@unt = \letter
    \fi
\else \s@ve \catcode \c@unt = \letter
\fi
\fi
\advance \c@unt by 1
\ifnum \c@unt < 128 % or 256 for the V3 sites ...
\repeat
\ch@rcode = '#1%                                %% we next save the character code of the
                                                %% character which has been specified as the
                                                %% escape character in \ch@rcode;
\edef \onvironment {%
    \onvironment \ch@rcode                         %% and append code to \onvironment
                                                %% to make the escape character active;
    \catcode \the \ch@rcode = \active
    \space                                         %% <space> separates list items in \onvironment
}%
\catcode \ch@rcode = \active                      %% the escape character is made active;
\uccode '\^M = \ch@rcode                         %% and the upper-case code of <return> is made
                                                %% equal to the character-code of the escape
                                                %% character; this is necessary because only
                                                %% <return> can be guaranteed to be active at
                                                %% this point, and we need an active character
                                                %% to form the primary operand of \def;
                                                %% the \exs below are \expandafters,
                                                %% and the effect is to upper-case <return>
                                                %% (yielding the escape character), then \def
                                                %% (an active instance of) this character as
                                                %% \n@sted, which has been defined above;

%
\@x \uppercase \@x {\@x \def \r@turn {\n@sted}}%
%
\uccode '\^M = 0                                 %% the upper-case code of <return> is then re-
                                                %% instated (not strictly true; it is set to 0,
                                                %% which is assumed to be its previous value
                                                %% — could be improved here);
                                                %% <return> is made active;
                                                %% and so is <space> (to avoid space-elision);
                                                %% finally, the finite-state automaton which
                                                %% processes <space>s is set to <void>;
                                                %% this ends the definition of \verbatim.

%
\def \s@ve #1#2=#3%
{\ifnum #1#2 = #3%

```

```

\else \edef \Onvironment
      {\Onvironment #1\the #2=\the #1#2 }%
#1#2 = #3%
\fi
}%

%%% the code which follows implements the finite
%%% state automaton which determines whether
%%% (space)s are ignored, shewn explicitly or
%%% implied, and which ensures that blank
%%% lines are reproduced correctly.

%
\def \v@id {\futurelet \n@xt \v@idifspace}%
\def \l@ad {\l@adingspace \futurelet \n@xt \l@adifspace}%
\def \sk@p {\vskip \baselineskip \futurelet \n@xt \l@adifspace}%
\def \emb@d {\emb@ddedspace}%
\def \sh@wspace {\char 32\relax}%
\def \h@despace {\leavevmode \kern \fontdimen 2 \font}%
\def \l@adingspace {\ifshewleadingspaces \sh@wspace \else \h@despace \fi}%
\def \emb@ddedspace {\ifshewembeddedspaces \sh@wspace \else \h@despace \fi}%
\def \v@idifspace {\testn@xt {\afterassignment \v@id}}%
\def \l@adifspace {\testn@xt {\afterassignment \sk@p}}%
%

%%% \testn@xt provides a common look-ahead for
%%% \v@idifspace and \l@adifspace, and also
%%% implements some essential debugging hooks.

\def \testn@xt #1%
{\ifx \n@xt \sp@c@
 \m@ssage {Next character is a space}%
 \let \n@xt = \relax
 \else \ifx \n@xt \r@t@rn
 \m@ssage {Next character is a return}%
 \def \n@xt {\#1\let \n@xt = }%
 \else \m@ssage {Next character is \meaning \n@xt}%
 \let \n@xt = \relax
 \o@x \let \sp@ce = \emb@d
 \fi
 \fi
 \n@xt
}%

%%%
\catcode '\ = \active%
\def\sp@ce{ }%
%%% We next tamper with the \catcode of (space)
%%% and (return), while defining macros and
%%% synonyms which require them to be active;
%%% the \catcode is then restored to its default
%%% (not necessarily the previous value —
%%% could be improved). \Octivespace makes
%%% (space) active, then defines (space) as
%%% \v@id with a synonym \sp@c@. This code is
%%% used by the finite-state automaton.

\def\Octivespace%
{\catcode'\ =\active\def {\v@id}\let\sp@c@= }\catcode'\ =10\relax%
%
\catcode '\^M = \active %
%%% (return) is made active;
\def \r@turn {\^M}%
%%% \r@turn defined as an active (return);
\let \r@t@rn = \^M%
%%% \r@t@rn is made a synonym;

```

```

\def \@ctivecr %           %%% and \@ctivecr is defined to
  {\catcode '\^M = \active % %% make <return> active, then
    \def ^M%                %% define <return> to manipulate the
                           %% finite-state automaton and ...
      {\@x \def \sp@ce {\l@ad}%
       \@x \let \@x \sp@c@ \@x =\sp@ce %
     }
%
\endgraf %               %% insert a \par primitive (for blank lines).
%
\futurelet \n@xt \l@adifspace %
}%
\let \r@t@rn = ^M%        %% \r@t@rn is synonymous with active <return>
}%
\catcode '\^M = 5 %       %% finally, the \catcode of <return> is
                           %% restored to its normal value;
%
%% the \AfterGroup macro is intended for
%% use within a nested normal environment,
%% and causes (a concealed macro defined as)
%% its parameter text to be inserted into
%% TeX's input stream when the nested normal
%% group terminates.
%
\def \AfterGroup #1{\global \def \@ftergroup {#1}\aftergroup \@ftergroup}%
%
\let \@x = \expandafter      %% \@x is a brief synonym for \expandafter;
%
\catcode '\@ = \other        %% commercial-at is restored to its normal
                           %% <other> catcode (not necessarily the
                           %% previous value — could be improved);
%
\def \mitabrev
  {\endgroup \endgroup}%
%
%% and \mitabrev defined as the closure for
%% \verbatim; any other name could be used,
%% as the code performs no look-ahead for
%% any particular string.
%
%% Finally we announce to the world that we
%% have been loaded, and give some clues as
%% to the usage.
%
\message {Verbatim environment loaded;}%
\message {usage: ``\noexpand \verbatim <char> ... <char> \noexpand \mitabrev''}%

```

◊ Philip Taylor  
 Royal Holloway and Bedford New College  
 P.Taylor@Vax.Rhbnc.Ac.Uk

## Easy Table

Khanh Ha

### Introduction

*Easy Table (EZ)* is an application tabular package designed to run independently of computer platforms. Its goals are to meet the most rigid requirements from trade typesetters when it comes to tabular work. EZ's refinements include: precision in row and column, vertical and horizontal placements; baseline and style control for horizontal rules; spanner headings; subspanner headings in four-level nestings; and a table splitting operation. EZ is a template-controlled program. It requires that table specifications be filled out once, and more important, it allows typing multi-line entries naturally anywhere in a table including in the spanner units. And table rows can be ended gracefully even with gutter rules but *without* the need to advance to the table's last column using &s.

Though a large program with about 200 commands, EZ is *easy* to learn, set up, and modify. Above all, it is comprehensive and highly precise.

If you master EZ, setting tables will be a joy, not a jolt. And when you reread Knuth's line, "Printers charge extra when you ask them to typeset tables, . . ." (*The T<sub>E</sub>Xbook*, p. 231), you might have a laugh or two.

### Table initialization with \tabinit

Because EZ relies on templates, it requires a proper table format specification before typesetting. Once a table format template is constructed, it is stored and can be reused later on. The \tabinit command is used to build a template for a table. This command specifies the following values:

1. Total columns
2. Table leading
3. Gutter width
4. Preambles

The template's general usage is:

```
\tabinit{<total columns>}{<tab leading>}{<gutter width>}{<preambles>}
```

Consider Table 1:

TABLE 1 Top 10 Newsstand Sellers (000s)

Magazine	1982	1986	% Change
1. TV Guide	9,732	8,234	-13.2
2. Family Circle	7,234	6,243	-15.4
3. Woman's Day	9,732	6,334	-11.2
4. National Enquirer	5,732	8,897	-23.2
5. The Star	9,732	4,833	-12.3
6. Penthouse	8,436	4,039	-43.4
7. Cosmopolitan	7,795	5,237	-22.5
8. Good Housekeeping	5,345	8,657	-16.7
9. People Weekly	7,322	7,342	-14.3
10. Globe	8,872	8,764	-11.7
	<b>80,022</b>	<b>68,580</b>	<b>-14.4</b>

Source: Statements of Ownership. Based on 1986 ranking.

CODES:

```
\tabinit{4}{10pt}{12pt}{\og{.5pt} % \og states outside gutter style
\c{1}{9pc}{l}{0pt}\c{2}{3pc}{c}{0pt}\c{3}{3pc}{c}{0pt}\c{4}{3pc}{c}{.5pt}}
```

```
\ninemoint
\ttitle{1}{Top 10 Newsstand Sellers (000s)}
\toprul
\hgstub{2em}
\tab{\bf Magazine\&\bf 1982\&\bf 1986\&\bf \% Change\et}
\tab{\en 1.\en TV Guide&9,732&8,234&$-13.2\et}
\tab{\en 2.\en Family Circle&7,234&6,243&$-15.4\et}
\tab{\en 3.\en Woman's Day&9,732&6,334&$-11.2\et}
\tab{\en 4.\en National Enquirer&5,732&8,897&$-23.2\et}
```

```

\tab{\en 5.\en The Star&9,732&4,833&$-12.3\et}
\tab{\en 6.\en Penthouse&8,436&4,039&$-43.4\et}
\tab{\en 7.\en Cosmopolitan&7,795&5,237&$-22.5\et}
\tab{\en 8.\en Good Housekeeping&5,345&8,657&$-16.7\et}
\tab{\en 9.\en People Weekly&7,322&7,342&$-14.3\et}
\tab{10.\en Globe&8,872&8,764&$-11.7\et}
\hrul{3pt}{0} % hrule clears outside rules
\tab{\bf 80,022\en&68,580\en&$-14.4\et}
\hrul{3pt}{0}
\tab{\bspan[1-4]} % 4-column body span
Source: Statements of Ownership. Based on 1986 ranking.\et}
\hrul{6pt}{1} % hrule joins outside rules

```

Calls to `\og` are of the form:

```
\og{\dimen}
```

The left outside gutter can be either a plain or ruled gutter. Use `\og{0pt}` if plain; if ruled, fill in units of measure for the rule weights by using one of the five different rule styles below:

`.5pt` = half point

`1pt` = 1 point

`2pt` = 2 points

`3pt` = 3 points

`99pt` = double rules

Calls to `\C` are of the form:

```
\C{\{col. "sequence\}}{\{col. "width\}}
{\{col. "justification\}}{\{gutter style\}}
```

`\C{\{col. sequence\}}` is the ordinal number of the column, from left to right. `\C{\{col. justification\}}` can have one of three values to determine a column's paragraph shape: `j=justified left/right`, `l=flush left/ragged right`, `r=flush right/ragged left`, and `c=ragged center`. `\C{\{gutter style\}}` is determined with values as discussed for `\og`. If the style is `0pt`, the gutter is blank; otherwise it has a vrule centered in it.

### Setting table entries with `\tab`

**EZ** builds a table by stacking its entries. It sets one entry after another with a command called `\tab`, ends an entry with another command called `\et`, and separates the columns in each tab field by the conventional `&`. Thus:

```
\tab{\{entry\}& . . . &\{entry\}\et}
```

The only command worth discussing here is `\et`.

### Ending a `\tab` entry with `\et`

The purpose of `\et` is to replace the primitive `\cr` while running **EZ**. `\et` ensures that all gutter rules in effect are output regardless of where the `\et` is

keyed. Needless to say, this helps reduce keystrokes in a multi-column table since stopping short to exit a row no longer requires `\&s` to advance to the end of a row. (Even if a table has no gutter rules, `\cr` is still unusable. As long as you rely on the template `\tabinit` you must use `\et`; bad output will result if you use `\cr`.)

### The table `hrules`

Horizontal rules are vital attributes in tables. Their role is to accent the main components by creating demarcations with their style and weight. Before setting a regular hrule three facts need to be determined: the rule's weight, leading, and style. A complete `\hrul` command has two parameters:

```
\hrul{\{leading\}}{\{style\}}
```

`\{leading\}` is the distance to the hrule's baseline measured from the base of the line above it; `\{style\}` is the way in which the hrule is drawn in relation to possible outside gutter rules. The value of `\{style\}` is an integer: a "0" will clear the hrule from outside rules; a "1" will connect the hrule with outside rules.

### Vertical alignment

Tables must cope with three different vertical alignments: top, bottom, and center. Alignments may be changed at any time by stating one of the commands `\aligntop`, `\alignbot`, `\aligncen`.

### Horizontal alignment

Each column style in **EZ**, justified or ragged, is determined by the `\C` command in the `\tabinit` template. Once this style is set, it becomes easier later on to concentrate on the data being entered. However, a need for a change in paragraph style while in a column necessitates a means to control

the paragraph shapes. To change the predetermined paragraph style while in a column one uses: \RR=ragged right; \RL=ragged left; \RC=ragged center; \XR=ragged cancelled (justified) within the appropriate field of a \tab.

### Vruled table

Tables with gutter rules should allow two features: 1) All gutter rules in effect should be drawn even in cases where rows end short; 2) Select gutter rules can temporarily be blank for special purposes. Of

these two requirements the first can be solved with the use of the command \et.

**The \et operation.** As previously discussed, the command \et is used to end a \tab row in the same fashion as the \cr. Fortunately, the \et is much more useful than \cr in the face of vrules. In fact, when vrules are present they will be all drawn automatically no matter where you end your row if and only if you use \et.

To learn about the usage of \et we have two illustrations below. First with rows completely filled with data in Table 2:

TABLE 2 The use of \et in normal situation

Depth Station	10m	25m	75m	100m	125m	150m	200m
3	0.73	0.76	0.37	0.08	0.02	0.06	-0.58
4	0.46	0.45	0.55	0.09	0.13	0.56	-0.76
5	0.78	0.43	0.67	0.11	0.21	0.08	-0.45
6	0.89	0.21	0.53	0.42	0.12	0.07	-0.15

CODES:

```
\ninepoint
\aligncen
\tabinit{8}{10pt}{12pt}{\og{.5pt}}
    \C{1}{5.5pc}{c}{.5pt}\C{2}{26pt}{c}{.5pt}\C{3}{26pt}{c}{.5pt}
    \C{4}{26pt}{c}{.5pt}\C{5}{26pt}{c}{.5pt}\C{6}{26pt}{c}{.5pt}
    \C{7}{26pt}{c}{.5pt}\C{8}{26pt}{c}{.5pt}

\ttitle{l}{The use of \et in normal situation}
\toprul
\tab{\RL\bf Depth\nl
    \RR\bf Station&
    \bf 10m&\bf 25m&\bf 75m&\bf 100m&\bf 125m&\bf 150m&\bf 200m\et}
\hrul{3pt}{1}
\tab{3&0.73&0.76&0.37&0.08&0.02&0.06&-$0.58\et}
\tab{4&0.46&0.45&0.55&0.09&0.13&0.56&-$0.76\et}
\tab{5&0.78&0.43&0.67&0.11&0.21&0.08&-$0.45\et}
\tab{6&0.89&0.21&0.53&0.42&0.12&0.07&-$0.15\et}
\hrul{3pt}{1}
```

Next with rows ending short in Table 3:

TABLE 3 The use of \et in special situation

Depth Station	10m	25m	75m	100m	125m	150m	200m
3	0.73	0.76	0.37	0.08	0.02	0.06	-0.58
4	0.46	0.45	0.55	0.09	0.13		
5	0.78	0.43	0.67	0.11	0.21	0.08	
6	0.89	0.21	0.53				

CODES:

```
\ninemoint
\aligncen
\restoretab{8} % bringing back the identical template
\ttitle{l}{The use of \et in special situation}
\toprul
\tab{\RL\bf Depth\nl
      \RR\bf Station&
      \bf 10m\&\bf 25m\&\bf 75m\&\bf 100m\&\bf 125m\&\bf 150m\&\bf 200m\et}
\hrul{3pt}{1}
\tab{3&0.73&0.76&0.37&0.08&0.02&0.06&-$0.58\et}
\tab{4&0.46&0.45&0.55&0.09&0.13\et}
\tab{5&0.78&0.43&0.67&0.11&0.21&0.08\et}
\tab{6&0.89&0.21&0.53\et}
\hrul{3pt}{1}
```

Thus far we have seen the usefulness of \et, especially in multicolumn vruled tables where one can exit a row gracefully, forgetting all the remaining tab alignments one would have needed to advance to reach the end of the current row. That leaves us with a second feature to explore: how to temporarily empty specific ruled gutters.

**Gutter rules temporarily blank with \bgut.** In order to void any gutter rules within a table body one must state a *range*, i.e., the starting column and

the ending column; this range includes the gutter rules to be blank:

\bgut[<start col>-<end col>]

where the hyphen (-) stands for possible columns in between, and the pair of brackets form the command's delimiters. This command can be stated either before or after the \tab starts; its effect will be limited to only the row it is issued for. Table 4 shows its usage:

TABLE 4 Blank Gutters

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

CODES:

```
\tabinit{6}{10pt}{12pt}{\og{.5pt}
  \C{1}{1pc}{c}{.5pt}\C{2}{1pc}{c}{.5pt}\C{3}{1pc}{c}{.5pt}
  \C{4}{1pc}{c}{.5pt}\C{5}{1pc}{c}{.5pt}\C{6}{1pc}{c}{.5pt}}
\ninemoint
\toprul
\ttitle{l}{Blank Gutters}
\toprul
\tab{\en 1\&\en 2\&\en 3\&\en 4\&\en 5\&\en 6\et}
\hrul{3pt}{1}
\bgut[1-3] % stated outside \tab
\tab{\en 7\&\en 8\&\en 9&10&11&12\et}
\hrul{3pt}{1}
\tab{13&14&15&16&17&18\et}
```

```
\hrule{3pt}{1}
\tab{\bgroup[4-6] 19&20&21&22&23&24\et} % stated inside \tab
\hrule{3pt}{1}
\tab{25&26&27&28&29&30\et}
\hrule{3pt}{1}
```

## Spanners

Spanners are tough customers. The first type of spanner is a column spanner, and the second a row spanner. Column spanners are entries that straddle a number of columns; they must be able to wrap around automatically in multi-line paragraph fashion with no contrived manual line-breaking operation; they must also be able to justify within their own territory in terms of raggedness or left and right justification; they ought to relate in vertical alignment to their possible counterparts on the same level in the row; their spanner rules must be flexible enough to remain within their own width or extend to join the neighboring gutter rules in addition to their rules' vertical adjustability for a particular leading; and they must be able to nest other spanners.

The second type of spanner is the row spanner. This kind is a column which spans vertically a number of rows and serves as their common heading.

**Body spanners with \bspan.** In a table body one might come across an entry that spans a number of columns. This type of column spanner must be specified by a *range*, i.e., the starting column and the ending column of the spanned columns. EZ's command for the body column spanner is \bspan, which is delimited by a pair of brackets:

```
\bspan[<start col>-<end col>]
```

\bspan must be stated *inside* the \tab command. Once the span is active, data can be poured into this space and EZ will handle the line-breaking algorithm in addition to the paragraph shape that has been specified. This shape is controlled by the style in the column that starts the body spanner, which has been specified in the \tabinit template. One can always override this style with one of the following commands: \JUST, \RR, \RL, \RC. Table 5 demonstrates the use of \bspan:

TABLE 5 Body Spanner

SUN	MON	TUES	WED	THUR	FRI	SAT
<i>Beat midsummer heat the cool and easy way with delightful, delicious dairy-fresh ice cream. Dip into the natural goodness of America's favorite treat. Build yourself the sundae of your dreams with your favorite fruits, nuts and syrups, or with the classic whipped cream and a cherry.</i>						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
<sup>23</sup> / <sub>30</sub>	<sup>24</sup> / <sub>31</sub>	25	26	27	28	29

CODES:

```
\ninepoint
\tabinit{7}{10pt}{12pt}{\og{.5pt}}
\c{1}{3pc}{1}{.5pt}\c{2}{3pc}{1}{.5pt}\c{3}{3pc}{1}{.5pt}
\c{4}{3pc}{1}{.5pt}\c{5}{3pc}{1}{.5pt}\c{6}{3pc}{1}{.5pt}
\c{7}{3pc}{1}{.5pt}
```

```

\toprul
\title{l}{Body Spanner}
\toprul
\bgut[1-7] % no gutter rules in the column headings
\tab{\bf SUN\&\bf MON\&\bf TUES\&\bf WED\&\bf THUR\&\bf FRI\&\bf SAT\et}
\hrul{6pt}{1}
\tab{\bspan[1-6]\it % ragged right is in effect from template for 2nd col.
      Beat midsummer heat the cool and easy way with delightful, delicious
      dairy-fresh ice cream. Dip into the natural goodness of America's
      favorite treat. Build yourself the sundae of your dreams with your
      favorite fruits, nuts and syrups, or with the classic whipped cream
      and a cherry.&&&&& % skip 6 spanned columns
      1\et}
\hrul{6pt}{1}
\tab{2&3&4&5&6&7&8\et}
\hrul{6pt}{1}
\tab{9&10&11&12&13&14&15\et}
\hrul{6pt}{1}
\tab{16&17&18&19&20&21&22\et}
\hrul{6pt}{1}
\tab{\frac{23}{30}\&\frac{24}{31}\&25&26&27&28&29\et}
\hrul{6pt}{1}

```

Because a body spanner always straddles a number of columns, it must collect an *equal* number of tab alignments from these spanned columns in order to move past them to the next column. Thus the spanner in this example needs six &s for six skipped columns. The paragraph shape of this spanner is controlled by the command "1" in the \tabinit template for the second column which starts the spanner. Again overriding the text shape of the spanner is simple, as already discussed.

**Head spanners with \spaninit.** The main difference between a body spanner and a head spanner is that the first acts like a header crossing a number of designated columns (it has no entries beneath it) while the second works as a header but also has entries under it and often has an hrule separating the header from its spanned entries. Therefore a head spanner must be treated as one complete unit that combines the header, the rule, and the spanned entries. To EZ each unit requires a column for itself; thus, for example, a 3-column spanner will occupy *one* single column and within this column the three spanned entries. EZ's method of setting a row with column spanners is to set each spanner

unit in a column completely, then move onward to the next possible column. It does *not* set across line for line as a typewriter would. Users need not guess entry depths nor make awkward attempts to break entries manually before setting the spanners. For clarity, EZ calls the head spanner *spanner head* and its spanned entries *spanner cells*; the separating rule between these two components is termed a *spanner rule*.

When a table has spanner heads, EZ uses this principle: the main body's template must be designed first before that of the spanner heads. Thus two templates will be present: 1) \tabinit for the main body, and 2) \spaninit for the column headings with spanner units.

**Spanner units in vruled tables.** The design for spanner units is the same in both table styles, vruled or plain. Spanner units which are ruled on all sides normally would incorporate sub-spanner units because these rules serve as demarcation. In plain tables the spanner units are single-level.

Now consider Table 6 for the introduction of spanner units:

TABLE 6 Vruled Spanner: Single Level

Size	Amps	Reserve capacity	Battery case size, inches			Catalog Number	Wt. lbs.
			L	W	H		
24°	500	115	10	6	8	28K 4315N	41
34°	502	116	20	7	8	28K 4314N	42
44°	504	117	30	8	8	28K 4313N	43
54°	506	118	40	9	8	28K 4312N	44
64°	508	119	50	2	8	28K 4311N	45
74°	510	120	60	3	8	28K 4319N	46

CODES:

```
\ninemoint
\aligncen
\tabinit{8}{10pt}{6pt}{\og{0pt} % 8 cols for main body
\c{1}{1.5pc}{1}{.5pt}\c{2}{2pc}{c}{.5pt}\c{3}{3pc}{c}{2pt}\c{4}{1pc}{c}{.5pt}
\c{5}{1pc}{c}{.5pt}\c{6}{1pc}{c}{2pt}\c{7}{4.5pc}{c}{.5pt}\c{8}{1.5pc}{c}{0pt}}}

\spaninit{6}{10pt}{6pt}{\og{0pt} % 6 cols for headings
\c{1}{1.5pc}{1}{.5pt}\c{2}{2pc}{c}{.5pt}\c{3}{3pc}{c}{2pt}
\c{4}{4pc}{c}{2pt}\c{5}{4.5pc}{c}{.5pt}\c{6}{1.5pc}{c}{0pt}}}

\def\dg{\circ} % degree symbol

\toprul
\ttitle{1}{Vruled Spanner: Single Level}
\toprul
\tab{Size&Amps&Reserve capacity&
\main{Battery case size, inches}{1} % Style 1: connecting gutter rules
\startmain
\maine{1pc}{L} | \maine{1pc}{W} | \maine{1pc}{H}
\endmain
&
Catalog Number& Wt. lbs.\et}
\hrul{3pt}{0}

\tab{24\dg&500&115&10&6&8&28K 4315N&41\et}
\tab{34\dg&502&116&20&7&8&28K 4314N&42\et}
\tab{44\dg&504&117&30&8&8&28K 4313N&43\et}
\tab{54\dg&506&118&40&9&8&28K 4312N&44\et}
\tab{64\dg&508&119&50&2&8&28K 4311N&45\et}
\tab{74\dg&510&120&60&3&8&28K 4319N&46\et}
\hrul{3pt}{0}
```

This table shows two templates, one for the main body, another for the column headings with spanner units. As already mentioned, the main body's template must be given first; for this table it prepares 8 columns. Next the template for the headings is designed for 6 columns because of

the 3-column spanner unit — this unit occupies one column. We turn now to the method of computing the total width of the spanner head.

**Setting the spanner heads with \main.** The command for spanner head is \main. Its form is:

```
\main{<Spanner Head Text>}{{Spanner Rule Style}}
```

where the spanner head text in the first argument can be multi-line data whose line-breaking algorithm is determined by the paragraph style commands, i.e., justified or ragged. The spanner rule style in the second argument controls the length of the rule: if "1", it will join the adjacent gutter rules; if "0", it will remain within the true width (the width of the spanner head).

**Setting the spanner cells with \maine.** Spanner cells are grouped together under the main spanner head. The adjective *main* is used to distinguish the first-level spanner head from the second-, third-, and fourth-level spanner heads. This complication arises when the column headings incorporate sub spanner units inside the main one. EZ allows four levels of spanners, descending from main to sub to subsub to subsubsub spanners. Regardless of the possible existence of the sub spanners, all entities under the main spanner head must start with \startmain and end with \endmain:

```
\main{\{spanner head\}\{\rule style\}}
\startmain
\maine{\{dimen\}\{\{cell text\}\} |
```

```
... \maine{\{dimen\}\{\{cell text\}\}}
\endmain
```

where \maine is a command for each spanner cell that looks for two arguments: 1) width of spanner cell, 2) text of spanner cell. The | will draw a gutter rule whose default weight is .5pt.

**Setting the spanner rules.** A spanner rule can have one of three styles specified by: "0" to stay within the true width of the spanner head; "1" to extend the rule to join adjacent, but not outside, gutter rules; "2" to extend rule to intersect all adjacent, including outside, gutter rules. Besides this option, the rule leading default (3pts) can be changed any time by stating

```
\sprskip{\{dimen\}}
```

where *dimen* is the leading amount for the spanner rule. The \sprskip command affects the leading change for the spanner rule only in the current column; thus changing leading for multiple spanner rules in different columns requires the \sprskip command to be restated each time in a new column. Making two changes to the rule leading and style of the previous example, we have:

Size	Amps	Reserve capacity	Battery case size, inches	Catalog Number	Wt. lbs.

#### CODES:

```
\ninemoint
\restorespan{6} % most recent spanner template
\toprul
\tab{Size&Amps&Reserve capacity&
\sprskip{6pt} % rule leading is now 6pt (good for one column only)
\main{Battery case size, inches}{0} % Style 0: not touching gutter rules
\startmain
\maine{1pc}{L} | \maine{1pc}{W}| \maine{1pc}{H}
\endmain
&
Catalog Number&Wt. lbs.\et}
\hrul{3pt}{0}
```

**Multiple spanners.** When a row has more than one spanner unit it must resolve an issue of alignment for the spanner heads and spanner cells. All

the spanner heads on a row will be related in their vertical alignment, and so are the spanner cells. Our first look at multiple spanners focuses on Table 7:

**TABLE 7** Multiple Spanners

Months and year	Number of stoppages		Workers involved		Days idle	
	Beginning in month or year	In effect during month	Beginning in month or year (in thousands)	In effect during month (in thousands)	Number (in thousands)	Percent of estimated working time
1947.....	270	.....	1,629	.....	25,720	—
1948.....	245	.....	1,435	.....	26,127	.22
1949.....	262	.....	2,537	.....	43,420	.38
1950.....	424	.....	1,698	.....	30,390	.26

CODES:

```

\aligncen
\ninepoint
\tabinit{7}{10pt}{6pt}{\og{.5pt}} % 7 columns for body
\C{1}{8pc}{1}{2pt}\C{2}{3.5pc}{c}{.5pt}\C{3}{3.5pc}{c}{2pt}\C{4}{3.5pc}{c}{.5pt}
\C{5}{3.5pc}{c}{2pt}\C{6}{3.5pc}{c}{.5pt}\C{7}{3.5pc}{c}{.5pt}

\spaninit{4}{10pt}{6pt}{\og{.5pt}} % 4 columns for headings
\C{1}{8pc}{1}{2pt}\C{2}{7.5pc}{c}{2pt}
\C{3}{7.5pc}{c}{2pt}\C{4}{7.5pc}{c}{.5pt}

\toprul
\ttitle{1}{Multiple Spanners}
\toprul

\tab{Months and year}
\main{Number of stoppages}{1}
\startmain
\maine{3.5pc}{Beginning in month or year} |
\maine{3.5pc}{In effect during month}
\endmain
&
\main{Workers involved}{1}
\startmain
\maine{3.5pc}{Beginning in month or year (in thousands)} |
\maine{3.5pc}{In effect during month (in thousands)}
\endmain
&
\main{Days idle}{2} % Style 2: intersecting outside rule
\startmain
\maine{3.5pc}{Number (in thousands)} |
\maine{3.5pc}{Percent of estimated working time}
\endmain

\et
\hrul{6pt}{1}

\tab{1947\dotlead&270\dotlead&1,629\dotlead&25,720\md\et}
\tab{1948\dotlead&245\dotlead&1,435\dotlead&26,127\&.22\et}
\tab{1949\dotlead&262\dotlead&2,537\dotlead&43,420\&.38\et}
\tab{1950\dotlead&424\dotlead&1,698\dotlead&30,390\&.26\et}

```

```
\hrule{6pt}{1}
```

Notice the correct vertical alignment of each component in all the spanner units: The spanner heads and also the spanner cells fulfill their assignments once they are able to relate vertically to each

other. To get a better look at this vertical relation, examine Table 8:

TABLE 8 Multiple Spanners

Warp. No.	Fabric Description	Thickness (in.)	Weight (oz/yd <sup>2</sup> )	Tensile Strength (lb/in.)		Tear Strength (lb)		Peel Strength (lb)		Flexural Rigidity* (lb/in. <sup>2</sup> /in.)	
				Warp	Fill	Warp	Fill	Warp	Fill	Warp	Fill

CODES:

```
\aligncen
\sixpoint
\tabinit{12}{8pt}{6pt}{\og{.5pt}}
\c{1}{1.5pc}{c}{2pt}\c{2}{7pc}{c}{2pt}\c{3}{3pc}{c}{2pt}\c{4}{3pc}{c}{2pt}
\c{5}{1.5pc}{c}{.5pt}\c{6}{1.5pc}{c}{2pt}\c{7}{1.5pc}{c}{.5pt}\c{8}{1.5pc}{c}{2pt}
\c{9}{1.5pc}{c}{.5pt}\c{10}{1.5pc}{c}{2pt}\c{11}{1.5pc}{c}{.5pt}\c{12}{1.5pc}{c}{.5pt}

\spaninit{8}{8pt}{6pt}{\og{.5pt}}
\c{1}{1.5pc}{c}{2pt}\c{2}{7pc}{c}{2pt}\c{3}{3pc}{c}{2pt}\c{4}{3pc}{c}{2pt}
\c{5}{3.5pc}{c}{2pt}\c{6}{3.5pc}{c}{2pt}\c{7}{3.5pc}{c}{2pt}\c{8}{3.5pc}{c}{.5pt}

\toprul
\ttitle{l}{Multiple Spanners}
\toprul

\tab{Warp. No.& Fabric Description& Thickness (in.)& Weight (oz/yd$^2$)&
\main{Tensile Strength (lb/in.)}{1}
\startmain
\maine{1.5pc}{Warp} |
\maine{1.5pc}{Fill}
\endmain
&
\main{Tear Strength (lb)}{1}
\startmain
\maine{1.5pc}{Warp} |
\maine{1.5pc}{Fill}
\endmain
&
\main{Peel Strength (lb)}{1}
\startmain
\maine{1.5pc}{Warp} |
\maine{1.5pc}{Fill}
\endmain
&
\main{Flexural Rigidity* (lb/in.$^2$/in.)}{2} % Style 2: intersecting outside rule
\startmain
\maine{1.5pc}{Warp} |
\maine{1.5pc}{Fill}
\endmain
\et}
```

```
\hrule{6pt}{1}
```

**Nested spanners.** Nested spanners occur when a spanner unit encloses another spanner unit which, in turn, comprises another one and so on. EZ supports four levels of nested spanners descending from the main to sub to subsub to subsubsub spanners. The commands for nested spanners are: \sub, \ssub, and \sssub, all below the main level of \main and restricted within the limit marked by \startmain and \endmain. All the subspanner units have the following usage:

```
\sub{\subspanner heading}{\rule style}{\subcell entries}
\ssub{\ssubspanner heading}{\rule style}{\ssubcell entries}
\ssub{\ssubspanner heading}{\rule style}{\ssubcell entries}
```

where the first two arguments are like those in \main. Only the third argument merits attention because

this extra argument makes \sub (and its counterparts) different from \main. A complete look at a subspanner unit:

```
\sub{\subspanner heading}{\rule style}{%
  % 3rd argument begins
  \sube{\cell width}{\entry text} |
```

```
  . . .
  \sube{\cell width}{\entry text}}
} % end of 3rd argument and subspanner
```

where \sube is much like \maine and | is the dividing rule between subspanner cells. The focus here is on the third argument for each subspanner unit: it must have matching braces to avoid "Runaway argument" errors. Spaces after the argument's open brace and also spaces before its close brace are ignored. To see how nested spanners work, examine Table 9:

TABLE 9 Nested Spanners

County	Industry	Employment change				Location quotient	
		Actual	Effect of shift-share by				
			State-wide	Industry mix	Country share		
Santa Cruz (Arizona):	Apparel..	7	28	48	32	5.9	

CODES:

```
\ninemoint
\aligncen
\tabinit{7}{10pt}{6pt}{\og{.5pt}} % 7 cols for body
  \C{1}{4pc}{1}{.5pt}\C{2}{4pc}{1}{2pt}\C{3}{3pc}{c}{.5pt}\C{4}{3pc}{c}{.5pt}
  \C{5}{3pc}{c}{.5pt}\C{6}{3pc}{c}{2pt}\C{7}{3pc}{c}{.5pt}

\spaninit{4}{10pt}{6pt}{\og{.5pt}} % 4 cols for headings
  \C{1}{4pc}{1}{.5pt}\C{2}{4pc}{1}{2pt}\C{3}{13.5pc}{c}{2pt}\C{4}{3pc}{c}{.5pt}

\toprul
\ttitle{l}{Nested Spanners}
\toprul

\tab{County& Industry&
  \main{Employment change}{1}
  \startmain
  \maine{3pc}{Actual} |
  \sub{Effect of shift-share by}{1}{ % subspanner cells go in here
```

```

\sube{3pc}{Statewide} |
\sube{3pc}{Industry mix} |
\sube{3pc}{Country share} % space ignored
} % end of subspanner

\endmain % end of main spanner
&
Location quotient\et}
\hrul{6pt}{1}

\tab{Santa Cruz (Arizona):\& Apparel\dotlead& 7& 28& 48& 32& 5.9\et}
\hrul{6pt}{1}

```

### More nested spanners

When spanners start to nest in multilevel fashion, the user is responsible for keeping track of each one, its start and its end. In practice it helps to type empty brace pairs first to lock up the design in

question, then start pouring in text data. It also helps to use a text editor which has a brace-checking utility to spot unbalanced braces. Table 10 features spanners with three levels deep:

TABLE 10 Nested Spanners

Year	Noninsti- tutional population	Labor force										Not in labor force	
		Employed				Civilian				Unemployed			
		Number	Percent of population	Total	Percent of population	Resident Armed Forces	Total	Agriculture	Nonagri- cultural industries	Number	Percent of labor force		

CODES:

```
\fivepoint
\aligncen
```

```

\tabinit{13}{.5pt}{\og{0pt}}
\c{1}{1.5pc}{1}{2pt}\c{2}{3pc}{c}{2pt}\c{3}{3pc}{c}{.5pt}\c{4}{3pc}{c}{.5pt}
\c{5}{1.5pc}{c}{.5pt}\c{6}{3pc}{c}{.5pt}\c{7}{3pc}{c}{.5pt}\c{8}{1.5pc}{c}{.5pt}
\c{9}{3pc}{c}{.5pt}\c{10}{3pc}{c}{.5pt}\c{11}{3pc}{c}{.5pt}
\c{12}{3pc}{c}{2pt}\c{13}{3pc}{c}{0pt}
```

```

\spaninit{4}{.5pt}{\og{0pt}}
\c{1}{1.5pc}{1}{2pt}\c{2}{3pc}{c}{2pt}\c{3}{28.5pc}{c}{2pt}\c{4}{3pc}{c}{0pt}}
```

```

\toprul
\ttitle{1}{Nested Spanners}
\toprul
```

```

\tab{%
  Year&
  Noninstitutional population&
  \main{Labor force}{1}
  \startmain
  \main{3pc}{Number} |
  \main{3pc}{Percent of population} |
  \sub{Employed}{1}{% subspanner begins (space ignored)
  \sube{1.5pc}{Total} |}
```

```

\sub{3pc}{Percent of population} |
\sub{3pc}{Resident Armed Forces} |
\ssub{Civilian}{1}{% subspanner begins
    \ssub{1.5pc}{Total} |
    \ssub{3pc}{Agriculture} |
    \ssub{3pc}{Nonagricultural industries}
} % end of subsub unit
} | % end of sub unit (| comes after sub ends)
\sub{Unemployed}{1}{% subspanner begins
    \sub{3pc}{Number} |
    \sub{3pc}{Percent of labor force} % space ignored
} % end of sub unit
\endmain % end of whole main unit
&Not in labor force\et}
\vskip-2pt
\hrul{3pt}{1}

```

**Spanners forever.** We close this section about column spanners with an example whose details in drawing nested spanners should be studied carefully

by readers to gain an idea about EZ's principles in spanner design:

TABLE 11 Heavy-Duty Spanners

Company and Market	Price						Volume		Earnings and Dividends				P/E Ratio	
	Last Week's Close	Pct. Change		5 Year		Last Week's		Earnings per Share			Indicated Dividend Yield	5 Year Average	Current	
		Last Week	4 Wks.	Year to Date	High	Low	Shares Traded	Shares Outstanding	Last 12 Months Amt.	5 Yr. Annual Growth Rate				
BaltGas	\$ 14.88	% .0	% 2.6	% .8	\$ 23.88	\$ 14.25	(000) 110	.51	\$ NA	% NA	% NA	% 12.2	— NA	

#### CODES:

```

\aligncen
\fivepoint
\tabinit{15}{6pt}{6pt}{\og{Opt}}
    \C{1}{3pc}{1}{.5pt}\C{2}{1.5pc}{r}{.5pt}\C{3}{1.5pc}{r}{.5pt}
    \C{4}{1.5pc}{r}{.5pt}\C{5}{1.5pc}{r}{.5pt}\C{6}{1.5pc}{r}{.5pt}
    \C{7}{1.5pc}{r}{.5pt}\C{8}{1.5pc}{r}{.5pt}\C{9}{2pc}{r}{.5pt}
    \C{10}{1.5pc}{r}{.5pt}\C{11}{2pc}{r}{.5pt}\C{12}{2pc}{r}{.5pt}
    \C{13}{1.5pc}{r}{.5pt}\C{14}{2pc}{r}{.5pt}\C{15}{2pc}{r}{0pt}

\spaninit{5}{6pt}{6pt}{\og{Opt}}
    \C{1}{3pc}{1}{.5pt}\C{2}{11.5pc}{c}{.5pt}
    \C{3}{4pc}{c}{.5pt}\C{4}{8.5pc}{c}{.5pt}\C{5}{4.5pc}{c}{0pt}

\toprul
\ttitle{1}{Heavy-Duty Spanners}
\toprul

\tab{Company and Market}
\main{Price}{1}
\startmain
\main{1.5pc}{Last Week's Close} | % 1st maine
\sub{Pct. Change}{1}{% } % 2nd maine a subunit
\ssub{Last}{1}{% } 1st sube is a ssub unit
    \ssub{1.5pc}{Week} |
    \ssub{1.5pc}{4 Wks.}

```

```

} | % gutrule MUST be given AFTER end of ssub
\sube{1.5pc}{Year to Date}
} | % end sub
\sub{5 Year}{1}{%      3rd maine a subunit
\sube{1.5pc}{High} |
\sube{1.5pc}{Low}
}
\endmain
&
\main{\bf Volume}{1}
\startmain
\sub{Last Week's}{1}{%
\sube{1.5pc}{Shares Traded} |
\sube{2pc}{Shares Out\standing}
} % end sub
\endmain
&
\main{\bf Earnings and Dividends}{1}
\startmain
\sub{Earnings per Share}{1}{%
\ssub{Last 12 Months}{1}{%
\ssube{1.5pc}{Amt.} |
\ssube{2pc}{Change}
} | % end ssub
\sube{2pc}{5 Yr. Annual Growth Rate}
} | % end sub
\maine{1.5pc}{Indi\cated Divi\end Yield}
\endmain
&
\main{P/E Ratio}{1}
\startmain
\maine{2pc}{5 Year Average} |
\maine{2pc}{Current}
\endmain
\et}
\vskip-1pt
\hrul{3pt}{0}

\tab{%
&\$&\%&\%&\$&\$(000)&\%&\$&\%&\%&\md&\md\et}
\tab{%
BaltGas& 14.88& .0& 2.6& .8& 23.88& 14.25& 110& .51&
NA& NA& NA& 12.2& NA& NA\et}
\hrul{6pt}{0}

```

**Row spanners.** In contrast to column spanners which cross columns, row spanners cross rows. The cell supposedly acting as a row spanner will vertically straddle a number of rows; in **EZ** such a spanner is achieved with **\xrow**. Its usage is:

\xrow{<entry text>}

and it is treated like a normal column. Table 12 provides an example:

TABLE 12 Row Spanners

<b>BATTERIES NOT INCLUDED. FULL 1-YEAR MONEY BACK GUARANTEE</b>	<b>Subtotal</b>	\$
	NY residents add sales tax	\$
	Add \$2 shipping no matter how many you order	\$
	<b>TOTAL</b>	\$

CODES:

```
\ninemepoint
\tabinit{3}{10pt}{6pt}{\og{.5pt}}
  \C{1}{7.5pc}{c}{.5pt}\C{2}{11.5pc}{r}{.5pt}\C{3}{3pc}{1}{.5pt}

\ttitle{1}{Row Spanners}
\toprul
\tab{\xrow{\bf BATTERIES NOT INCLUDED. FULL 1-YEAR MONEY BACK GUARANTEE}
  & Subtotal&$\et}
\prul[2-3,6pt] % partial rule for 2-3 cols; on 6pt leading
\blank{2pt}
\tab{& NY residents add sales tax&$\et}
\prul[2-3,6pt]
\blank{2pt}
\tab{& Add \$2 shipping no matter how many you order&$\et}
\hrul{6pt}{1}
\blank{2pt}
\tab{&\RL TOTAL&$\et}
\hrul{6pt}{1}
```

Note that the row spanner acts like a normal column because, after setting it, one must tab across to the next column. While working with a row spanner, \prul will be needed to draw partial hrules which avoid the cell the row spanner occupies. Once beyond the row spanner, the normal \hrul for full-width hrules can be used.

**Conclusion.** This article is excerpted from the 118-page operation manual of *Easy Table*. Its purpose is to illustrate a few major features of this software. For more information about the purchase of *Easy Table* software, please contact me at 301-598-0557, or write to:

- ◊ Khanh Ha  
14912 Village Gate Drive  
Silver Spring, MD 20906

---

## Typesetting Bridge via $\text{\TeX}^1$

Kees van der Laan

### Abstract

Enhanced plain  $\text{\TeX}$  macros and a bidding environment for typesetting bridge card distributions and bidding sequences are given as a follow-up to the  $\text{\LaTeX}$  macros given in [12]. Moreover, macros for annotated printing of the course of the play are provided. Examples of use are included.

### Introduction

After the publication of [12], Bernard Gaulle among others, asked for plain  $\text{\TeX}$  macros with similar capabilities. This article concentrates on

- a. Translation into plain  $\text{\TeX}$  of the  $\text{\LaTeX}$  macros for printing card deals and bidding sequences as published in [12], i.e., emulated  $\text{\hand}$ ,  $\text{\crdima}$  macros and a NESW-figure, as well as a flexible ( $\text{\bbid}$ ,  $\text{\ebid}$ ) environment.
- b. (new)  $\text{\TeX}$  macros —( $\text{\bplay}$ ,  $\text{\eplay}$ ) environment and  $\text{\showgame}$ — for handling the course of the play. These macros imitate the spirit in which chess is ‘played’ in print (i.e., with annotations and preserved data-integrity; see [2, 16]), no retying of the hands! Discussion starts in the section “How the play goes.”

The translated macros are enhanced with respect to both language and application flexibility. The language flexibility is in the spirit of the ‘international’ DUTCH-sty-option activity (see [4]). Names are provided, via (grouped) macros, which can be redefined easily. Within the context of bridge this means redefinition of the four hands

```
\def\FIH{North}%
\def\SEH{East}%
\def\THH{South}%
\def\FOH{West}%
```

and redefinition of  $\text{\N}$ ,  $\text{\E}$   $\text{\S}$ ,  $\text{\W}$ ,  $\text{\EW}$ ,  $\text{\NS}$ ,  $\text{\TRICK}$ .

---

<sup>1</sup> Presented at GUTenberg 90.

*Note added in proof:* This paper has been improved with respect to the earlier GUT’90 version, GUTenberg cahiers 5. The improvements are: inclusion of explicit  $\text{\english}$ ,  $\text{\french}$  and  $\text{\dutch}$  commands; no separate ( $\text{\bbidcmp}$ ,  $\text{\ebidcmp}$ )-environment is needed; and, in printing, the course of a play each trick starts with the lead.

In several books, e.g. [13], the players are personalized into: Partner, RHO, YOU, LHO, where R/L-HO mean Right/Left-Hand Opponent. In newspaper columns the names of the players are sometimes given. This, as well as language variations, can be realized easily by redefinitions of  $\text{\FIH}$ , etc. It must be admitted, though, that editing source texts is in general not that difficult, just cumbersome.

As long as card values are represented by digits and letters we don’t need control sequences for them. They can just be typed in, with the representation you like. We have A(ce), K(ing), Q(ueen) and J(ack), in English; A(s), R(oi), D(ame), V(alet), in French; while in Dutch they read A(as), H(eer), V(rouw), B(oer); along with T(en) —respectively T(ien), or generally 10— 9, 8, 7, 6, 5, 4, 3, 2.

### Card deals

$\text{\hand}$  prints the cards a player holds.  $\text{\crdima}$  (CaRD IMAge) prints the cards given for all four hands in a suitable way. The argument sequences of  $\text{\hand}$  and  $\text{\crdima}$  are similar to the  $\text{\LaTeX}$  argument sequences given in [12].

**Arguments.**  $\text{\crdima}$  takes six arguments:

**first argument:** text. In particular, this argument specifies the dealer and the vulnerability. For example: N/None means North dealer and vulnerability none.

**second argument:** text. For example, indication of deal as in Deal 1 or in

```
\vtop{\hbox{Deal:}%
      \hbox{demo }}
```

**next four arguments:** the four hands N, E, S, W, clockwise. Each hand is a call of the  $\text{\hand}$  macro with four arguments: the ♠, ♥, ♦, ♣ cards.

The central figure is contained in a box register,  $\text{\NESW}$ .

For example,

```
$$\text{\crdima}{N/None}{\vtop{\hbox{Deal:}%
      \hbox{demo}}}{%
  {\hand{J74}{AJ}{QJT2}{Q874}}{%
  {\hand{K86}{T9542}{874}{T3}}}{%
  {\hand{QT952}{Q83}{AK5}{A6}}}{%
  {\hand{A3}{K76}{963}{KJ952}}}{%
  $$
```

yields

N/None	♠ J74 ♥ AJ ♦ QJT2 ♣ Q874	Deal: demo
♠ A3 ♥ K76 ♦ 963 ♣ KJ952	♠ K86 ♥ T9542 ♦ 874 ♣ T3  ♠ QT952 ♥ Q83 ♦ AK5 ♣ A6	

### Bidding

The bidding environment is not based on tabbing; `\halign` is used. This means that the bidding sequences are lines within `\halign`, with four columns, and have to obey the usual alignment syntax. The card deal above takes the following ACOL bidding

North	East	South	West
1♣ <sup>A</sup>	? no	1♠	...no
2♠	no	4♠	a.p.

<sup>A</sup> means Alert, conventional bid  
? means explanation asked  
... means think pause

obtained via

```
{\medskip\narrower\noindent
\bbid
1\c\alert& ? no& 1\s&\think no\cr
2\s& no& 4\s& a.p.\cr
\noalign{\vskip.5ex}
\alert\ means Alert,
conventional bid\hidewidth\cr
? means explanation
asked\hidewidth\cr
\think means think
pause\hidewidth\cr
\ebid \medskip}
```

**Remarks.** One has to have a nodding knowledge of **TEX**. A more user-friendly `\annotation` command can be written, in the same spirit as a footnote or endnote.<sup>1</sup>

<sup>1</sup> A simple approach could be a command with two arguments where the first argument contains the annotation symbol(s), the second argument contains the explanation, and both are passed on to control sequences (or token registers). `\ebid` must be redefined so that the annotation(s) will appear.

Another issue is whether we should test for illegal biddings. I did not do this because it will restrict the use of the macros—illegal biddings are needed in arbiter courseware, for example.

The above is natural and will suffice for simple applications. The given `\crdima` and `\hand` macros as well as the bidding environment can be used in a way similar to their **LATEX** predecessors. So ‘drivers’—e.g., my (Pascal) deal program, for prints of tournament plays—hardly need to be adapted.

Furthermore, **LATEX** users can also make use of these enhanced versions at the expense of `\halign`’s syntax for the bid sequences.

In order to handle other bridge typesetting usages<sup>2</sup> elegantly and consistently, we have to think more thoroughly about how to pass information from one macro to another.

### Variables and parameters vs. control sequences and arguments

Knuth, [11, p.211], names the possibilities:

“It is sometimes desirable to pass information from one macro to another, and there are several ways to do this: by passing it as an argument, by putting it into a register, or by defining a control sequence that contains the information.”

It is not clear to me what to provide via arguments, what via registers and what via control sequences from one macro to another. The above is the **TEX** terminology and well-defined, while in Pascal-like programming we call the possibilities:

- transfer via parameters (by name, reference or value),
- via global variables, and
- via procedures.<sup>3</sup>

In command languages (and also in ADA) we distinguish between parameters bound to a position and bound via keywords in free order along with defaults.

In `\crdima` the texts and hands, and in `\hand` the cards for every colour are provided via arguments. Another approach is to provide all this information via control sequences, i.e., control sequences for

- the vulnerability and dealer information,

<sup>2</sup> In practice simpler techniques are used: Meulenbroek, for example, edits the previous column with the word processor at hand.

<sup>3</sup> In numerical mathematics we also have what is called reverse communication.

```
\def\LFTINF{N/None}%
- general information,
\def\RGTINF{Demo}%
- cards per colour and player, i.e., \Ns, for
North's ♠'s, etc.
```

One could then introduce something like `\showgame`, with *no* arguments, which uses these control sequences. This is done in the section on "How the play goes."

So, there is essentially one 'variable' left, the representation of the NESW-figure. One could use the optional parameter mechanism (see e.g. [3]) with the disadvantage that this parameter must be supplied for every deal once a personalized layout, different from the default, has been chosen. In my opinion this kind of variability, which is no longer there once personalized, can best be provided via a register, e.g., a box register in this case, and not via an optional parameter. When no figure is wanted, just 'empty the box', and when you would like one of your own use `\setbox\NESW\hbox{...}`. The notation for the players used in the NESW-figure is contained in control sequences, `\N`, etc.

In the bidding environment the notation for the players is also contained in control sequences, `\FIH`, etc. This provides language as well as order flexibility. Annotation commands are, e.g., `\alert`, `\think` (think pause), ? (before the bid: explanation is asked for; after the bid: questionable bid), whatever you like to add, and various combinations, such as question followed by think pause.

In the play environment the lead can be specified by `\LEADN`, `\LEADE`, `\LEADS`, or `\LEADW`. These control sequences set the definitions of `\FIP%First Player`, `\SEP`, `\THP`, and `\FOP`. Furthermore, the cards played have to be given in (English) natural notation, e.g., h8 for ♡8. The `\bintermezzo ... \eintermezzo` environment is a more user-oriented disguise for `\noalign`.

**Remark.** It is tempting to ponder about where keyword parameters come in (see e.g., [1]). Think of modifying the contents of a register or redefining a control sequence. The functionality is already there—for example, see the section on application flexibility.

### Notation

For the names of the control sequences for the hands and the left and right information, I adopted upper case letters `\FIH`, `\SEH`, `\THH`, `\FOH`; `\N`, `\E`, `\S`, `\W`, `\NS`, `\EW`; `\LFTINF`, `\RGTINF`, and for the colours of the cards and for the annotation

commands I used lower case letters `\s`, `\h`, `\d`, `\c`; `\alert`, `\think`. For the lead indication and `F`irst, `S`econd, etc. Player I also used upper case letters: `\LEADN`, `\LEADE`, `\LEADS`, `\LEADW`; `\FIP`, `\SEP`, `\THP`, `\FOP`. Language commands are also in lower case; supplied are `\english` (default), `\dutch`, and `\french`. This naming convention also holds for name combinations in the control sequences for the cards per hand per colour, i.e., `\Ns`, etc. Note that we have `\NS` and `\Ns`, denoting respectively the North-South combination and North's ♠'s.

**Remark.** With respect to choosing another language, I adopted that the result *in print* will be in the specified language; the control sequences remain in English. Data which will be printed—card values—have also to be supplied in the other language. Note that the card *colours* have to be denoted in English: ♦'s are always denoted by `h` (in play environment) or `\h` (in bidding environment).

### Application flexibility

a. **Another language.** In the following, the French language is used.

N/Personne	♠ V74 ♡ AV ♦ DV102 ♣ D874	♠ R86 ♡ 109542 ♦ 874 ♣ 103
♠ A3 ♡ R76 ♦ 963 ♣ RV952	♠ D10952 ♡ D83 ♦ AR5 ♣ A6	

takes the following ACOL bidding

Nord	Est	Sud	Ouest
1♣ <sup>A</sup>	pas	1♠	... pas
2♠	pas	4♦	pas
pas	pas		

obtained via

```
{% Local change,
\french
{\medskip\narrower\noindent
\crdima{N/Personne}{}
{\hand{V74}{AV}{DV102}{D874}} %N
{\hand{R86}{109542}{874}{103}} %E
{\hand{D10952}{D83}{AR5}{A6}} %S
{\hand{A3}{R76}{963}{RV952}} %O
\medskip}
\noindent takes the following ACOL
```

```

bidding
{\medskip\narrower\noindent
\bbid
  1\c\alert& pas& 1\s& \think pas\cr
    2\s& pas& 4\s&      pas\cr
      pas& pas\cr
\ebid          \medskip
}\% end local change

```

**b. Changing order.** If for some reason one likes to start with another player, e.g. West, in the printing of the bidding sequences, with the same dealer and vulnerability, this yields

West	North	East	South
-	1♣ <sup>A</sup>	no	1♣
...no	2♠	no	4♦
a.p.			

and is obtained via

```

\% Local change, note that the order
% of the defs is free
\def\FIH{West}\def\SEH{North}
\def\THH{East}\def\FOH{South}
%
\medskip\narrower\noindent
\bbid
  --& 1\c\alert& no& 1\s\cr
\think no& 2\s      & no& 4\s\cr
  a.p.\cr
\ebid          \medskip
}\% end local change

```

Another adaption is using a different naming, e.g., first hand is Partner via \def\FIH{Partner} etc. See the section on Endplay analysis, where \N, etc., are personalized.<sup>3</sup>

**c. Natural notation for input.** Natural notation is bound to a language. This gives complications if one likes to specify the card colours. For example in the French language we have carreaux and coeurs, which both abbreviate to c.

Furthermore, one can think of hiding TeXnicalities. The latter means that one could omit & and \cr and use, respectively, a space and a carriage return instead. I decided not to hide & and \cr.

One can also think of denoting the colours via the first character of the colour names in the bid

---

<sup>3</sup> This modification can be simplified when the NESW-figure is not put in a register, i.e., \def\NESW{\hbox{\NESWfig}} and \\$\vcenter{\NESW\\$} are used.

environment instead of the corresponding control sequence. I decided to have control sequences in the bid environment for the colours, because this makes it possible to supply any prefix. In the play environment I decided in favour of the colour abbreviation, s, h, d, or c, because there is no need for prefixes.

**Remarks.** Note the keyword functionality in examples a and b.

The general disadvantage of flexibility is the need for discipline; no consistency is forced. The advantage is freedom, and the question is how to use it.

### Macro texts

The provided NESW-figure is implemented via a 'ruled' table. The N, E, S, W symbols are provided via control sequences. The positioning obeys \halign's rules.

**Source texts.** \hand, \crdima, \NESW, and (\bbid, \ebid)

```

\def\hand#1#2#3#4{%
  %Example: \hand{AKJ765}{AK9}{--}{T893}
  \vtop{\hbox{\strut\s\enspace#1}
    \hbox{\strut\h\enspace#2}
    \hbox{\strut\d\enspace#3}
    \hbox{\strut\c\enspace#4}}%end \vtop
}%end \hand

\def\crdima#1#2#3#4#5#6{%
  %purpose: layout bridge hand
  %#1 left upper text
  %#2 right upper text
  %#3, #4, #5, #6: N, E, S, W hands
  \vbox{\halign{####\quad\cr
    #1& #3& #2\cr
    $ \vcenter{#6} $$ \vcenter{\copy\NESW} $&
      $ \vcenter{#4} $ \cr
    & #5& \cr
  }%end \halign
}%end \vbox
}%end \crdima

\def\NESWfig{%
  \vbox{\font\small=cmr9
    \def\str{\vrule height2.2ex%
      depth.75ex width 0pt}
    \offinterlineskip\tabskip0pt\hrule
    \halign{\vrule\hskip2pt\relax
      ##\hfil\tabskip3pt&
        \str\hfil##\hfil&
    }
  }
}
```

```

##\hskip2pt\relax\hfil\vrule
  \tabskip0pt\cr
  & \hbox to Opt{\hss\N\hss}\& \cr
\W&           \phantom{N}\& E\cr
  &\str\hbox to Opt{\hss\S\hss}\&\cr
  }%end \halign
  \hrule\%end \vbox
}% end \NESWfig
\setbox\NESW\hbox{\NESWfig}

\def\ebid{\errormessage{%
  bbit command is missing}

\def\bbid{\bgroup
  \def\ebid{\egroup\egroup\egroup}
  \def\alert{$^A$}
  \def\think{$\ldots$\thinspace}
  % etc.
  \vtop\bgroup
  \halign to4\wr\bgroup\tabskip3ex
    plus 1ex minus 1ex\hfil\cr
    \FIH\hfil&\SEH\hfil&
    \THH\hfil&\FOH\hfil\cr
  }%end \bbid

```

**Remark.** plain TeX macros for nicely rounded frames, LATEX's 'ovals', have been published (see [8]). They can be used for another frame representation in NESW.

### Some more examples

a. In order to illustrate general bidding theory from the viewpoint of one hand only, the \hand macro can be used. The following layout, heavily used in [7],

♠ AKJ42	North	East	South	West
♥ AK9	1♠	no	1NT	2♣
♦ T832	?			
♣ T				

is obtained via

```

{\medskip\narrower
\hbox to \hsize{\hss
\hand{AKJ42}{AK9}{T832}{T}%
\quad\hfil
\bbid
1\s& no& 1NT& 2\c\cr
?\cr
\ebid
\hss} \medskip}

```

b. For issues related to defense play one often displays only the dummy hand and your own hand. The following example is borrowed from [5].

♠ AJ632	N
♥ 43	W E
♦ KQ7	
♣ A85	You
♠ 985	
♥ 852	
♦ AJ5	
♣ KQT3	

North	East	South	West
-	-	-	1♠
no	2♥	no	2NT
no	4♥	a.p.	

Against 4♥ South starts ♣K, taken with ♣A. Leader continues ♥AKQ. On the third round of ♥'s, partner discards ♦9 (indicates interest in ♠). Leader continues with ♦2, how do you continue?

The example is obtained via

```

{\def\$You\% local change
\setbox\NESW\hbox{\NESWfig}
\medskip\narrower\noindent
\crdima{}%
{}-\{\hand{985}{852}{AJ5}{KQT3}\}%S
{}-\{\hand{AJ632}{43}{KQ7}{A85}\}%W
\medskip
} end local change NESW-figure

```

```

{\medskip\narrower\noindent
\bbid
--& --& 1\s\cr
no& 2\h& no& 2NT\cr
no& 4\h& a.p.\cr
\ebid \medskip}

```

**Remark.** In a similar way W-N, N-E, E-S hands, or W-E, N-S hands, or one hand only, with NESW-diagram, can be displayed simply by a suitable call of \crdima.

c. In discussing endplays only a few cards are left. The following endplay is taken from [10].

♠ AJ	S leads ♣A,
♥ K	W is squeezed
♦ -	
♣ -	
♠ KQ	
♥ A	N W E S
♦ -	
♣ -	
♠ 7	7
♥ 9	
♦ T	
♣ -	
♠ 2	
♥ 4	
♦ -	
♣ A	

The example is obtained via

```
\medskip\narrower\noindent
\crdimaf{}\vtop{\hbox{S leads \c A,}}
\hbox{W is squeezed}}\%
\hand{AJ}{K}{--}{--}\%N
\hand{7}{9}{T}{--}\%E
\hand{2}{4}{--}{A}\%S
\hand{KQ}{A}{--}{--}\%W
\medskip
```

d. Finally, a bidding competition. It illustrates how the (`\bbid`, `\ebid`) environment can be used for this application. We have taken only two partnerships: Sjoerd&Martijn and Tsjip&Janski. The material is borrowed from [17].<sup>4</sup>

W/All; Bidding competition

♠ AJ8	♠ -
♥ AKT94	♥ J8
♦ 8	♦ AKQ54
♣ KT98	♣ AJ7543

On the above hands, and given that South will intervene with 4♠, the partnerships bid as follows,

West	East	West	East
Sjoerd	Martijn	Tsjip	Janski
1♥	2♣	1♥	2♦
(4♠ by South)		(4♠ by South)	
no <sup>1</sup>	5♣ <sup>2</sup>	dbl	6♣
7♣	no	no	

<sup>1</sup> Forcing pass

<sup>2</sup> Grand slam try

obtained via

```
$$\crdimaf{ W/All;
Bidding competition \hidewidth\cr
\noalign{\vskip.5ex}}\%
{\\hand{--}{J8}{AKQ54}{AJ7543}}\%E
```

<sup>4</sup> Normally, the set of West-hands is separated from the set of East-hands.

```
\{\hand{AJ8}{AKT94}{8}{KT98}} \%W
$$
\noindent On the above hands, and given
that South will intervene with 4\s,
the partnerships bid as follows,
%
{\medskip\narrower
\hbox to \hsize{\hss
\%Sjoerd&Martijn (Local mods)
\def\FIH{\vtop{\hbox{West}
\hbox{\it Sjoerd\}/}}}
\def\THH{\vtop{\hbox{East}
\hbox{\it Martijn\}/}}}
\def\SEH{}\def\FOH{}
\def\bidwidth{3\wr}
\bbid
1\h& &2\c\cr
(4\s by South)\hidewidth\cr
no\$^1\$ &5\s\$^2\$ \cr
7\c& &no\cr
\noalign{\vskip.5ex}
\$^1\$ Forcing pass\hidewidth\cr
\$^2\$ Grand slam try\hidewidth\cr
\ebid\%end Sjoerd&Martijn
\quad
\%Tsjip&Janski (Local mods)
\def\FIH{\vtop{\hbox{West}
\hbox{\it Tsjip\}/}}}
\def\THH{\vtop{\hbox{East}
\hbox{\it Janski\}/}}}
\def\SEH{}\def\FOH{}
\def\bidwidth{3\wr}
\bbid
1\h& &2\d\cr
(4\s by South)\hidewidth\cr
dbl& &6\c\cr
no\cr
\ebid\%end Tsjip&Janski
\hss\%end \hbox
\medskip}
```

### How the play goes

Explanatory schemes of a play are used for instance on viewgraphs instantly during a match, in books about play technique, or in newspaper columns when discussing interesting matches or puzzles. In order to do this systematically and unambiguously, something similar to the 'algebraic' notation in chess (see [2, 16]) is needed.

Agreed, reading a book filled mostly with (algebraic) notation tables is quite dull and we can never replace the literarily gifted commentator. So,

this reduces the practical value of the exercise, but for solutions of puzzles it might be quite efficient, although I don't expect that many solutions will be sent in using TeX, in spite of quite numerous bridge unions, e.g., NBB (75,000 members), [5], to name but one union. On the other hand, the systematic approach eliminates misprints in shown phases while discussing a play.

Anyhow, it was great fun, and I learned a lot from it.

What we need is a compact unambiguous notation which contains per trick the information about the cards played and who led. Who gained the trick<sup>5</sup> can be deduced from the general knowledge of the contract and the lead. In print one generally starts every trick with the lead; every card that is played is given by the card colour and card value, followed eventually by commentary symbols like !, or ?.

To print all this information, I used basically a table with four columns (the players) and thirteen rows (the tricks). Each row starts with the lead.<sup>6</sup> Apart from printing the cards played (along with trick number), the cards in every hand — the (toks register) control sequences \Ns, etc. — are updated. The use is illustrated below.

### Let us play a game

The following appeared in 'Meulenbroek's column' last Christmas.<sup>7</sup>

Puzzle	<p>♠ KQ76    6NT,       ♥ J98    by East       ♦ J942       ♣ 65</p> <table border="1" style="margin-left: 20px;"> <tr><td>♠ AJ3</td><td>♠ T9</td></tr> <tr><td>♥ K653</td><td>♥ A2</td></tr> <tr><td>♦ AK3</td><td>♦ T5</td></tr> <tr><td>♣ AQT</td><td>♣ KJ9xxxx</td></tr> </table> <p>♠ 8542 ♥ QT74 ♦ Q876 ♣ 2</p>	♠ AJ3	♠ T9	♥ K653	♥ A2	♦ AK3	♦ T5	♣ AQT	♣ KJ9xxxx
♠ AJ3	♠ T9								
♥ K653	♥ A2								
♦ AK3	♦ T5								
♣ AQT	♣ KJ9xxxx								

**Problem.** How must NS defend in order to guarantee 1 trick?

<sup>5</sup> On viewgraphs underlining is commonly used; this can be implemented, but because of entailed inflexibility I refrained from it.

<sup>6</sup> The lead indication can be hidden for the first lead in something like \contract, \leader or explicitly \lead, and for the next tricks along with the automation of who gained the trick.

<sup>7</sup> Borrowed from [6].

**Solution.** Start with a ♠ lead in order to break communication. N must discard ♠'s and S must discard ♣'s.

Trick	NS	EW
1.	♥ 4!	♥ K
2.	♣ A	♣ 5
3.	♣ Q	♣ 6
4.	♣ T	♥ 9
5.	♣ J	♠ 5
6.	♣ 9	♠ 8
7.	♣ x	♦ 6

On lead of the next ♣ neither South nor North can be squeezed as can be seen from

Puzzle	♠ KQ ♥ J ♦ J94 ♣ ~	NS squeezed on ♣ continuation?
	♠ A	♠ T9
	♥ 63	♥ A
	♦ AK3	♦ T5
	♣ -	♣ x
	♠ -	
	♥ QT7	
	♦ Q87	
	♣ -	

with continuation

8.	♣ x	♥ 7	♥ 6	♥ J	-	8
9.	♦ T	♦ 7	♦ A	♦ 4	-	9
10.	♦ K	♦ 9	♦ 5	♦ 8	-	10
11.	♥ 3	♦ J	♥ A	♥ T	-	11
12.	♠ T	♥ Q	♠ A	♠ Q	-	12
13.	♦ 3	♠ K	♠ 9	♦ Q	1	12

**Input.** The above is obtained by

```
\def\LFTINF{Puzzle}
\def\RGTINF{\vtop{\hbox{6NT,}
\hbox{by East}}}

%
\Ns={KQ76}\Es={T9}\Ss={8542}\Ws={AJ3}
\nh={J98} \eh={A2}\sh={QT74}\wh={K653}
\nd={J942}\ed={T5}\sd={Q876}\wd={AK3}
\nC={65}\Ec={KJ9xxxx}\Sc={2}\Wc={AQT}
%
\showgame
%
\subhead *Problem*
How must NS defend in order to
guarantee 1 trick?
%
\subhead *Solution* Start with a \h\
lead in order to break communication.
N must discard \h's
```

```

and S must discard \s's.
\smallskip\noindent
\LEADS
\bplay
h4! & hK & h8 & h2 & -- & 1\LEADW\cr
cA & c5 & cx & c2 & -- & 2\cr
cQ & c6 & cx & s2 & -- & 3\cr
cT & h9 & cK & s4 & -- & 4\LEADE\cr
cJ & s5 & s3 & s6 & -- & 5\cr
c9 & s8 & h5 & s7 & -- & 6\cr
cx & d6 & sJ & d2 & -- & 7\cr
\bintermezzo
On lead of the next \c\
neither South nor North can be
squeezed as can be seen from%
\def\RGTINF{\vtop{\hbox{NS squeezed on}
\hbox{\c continuation?}}}
\showgame
with continuation
\eintermezzo
cx & h7 & h6 & hJ & -- & 8\cr
dT & d7 & dA & d4 & -- & 9\LEADW\cr
dK & d9 & d5 & d8 & -- & 10\cr
h3 & dJ & hA & hT & -- & 11\LEADE\cr
sT & hQ & sA & sQ & -- & 12\LEADW\cr
d3 & sK & s9 & dQ & 1 & 12\cr
\eplay

```

**Remark.** The cumulative tricks can be suppressed by deleting columns 5 and 6 and emptying the head texts via \def\NS{} and \def\EW{}.

#### Macros for annotated play

The (\bplay, \eplay) environment is aimed at printing schematically the cards played. Interleaving remarks, showing the phase of the play etc., can be supplied within the

```
\bintermezzo ... \eintermezzo
```

subenvironment. \pc does two things: it prints the card played and deletes the card from the appropriate hand. \strip essentially strips out one symbol from a string. \showgame is just a call of \crdima with the current values of \Ns etc.

**Explanation.** The problem is to determine dynamically with which colour from which player we are dealing. In each column of \bplay the player is known via the control sequences \FIP, \SEP, \THP and \FOP (these are eventually adjusted by \LEADN, \LEADE, \LEADS, or \LEADW) and passed on to \pc, as first argument (see template line of \halign in

\bplay). From the typed-in information, within the (\bplay, \eplay) environment, the colour is passed on as second argument to \pc. Symbols after that are handled as text, and influence \halign's column positioning.<sup>8</sup> \strip is called by \pc to delete a symbol. The symbol that has to be located in the string is used as argument separator.

#### Source texts.

```

\def\eplay{\errormessage{%
\bplay command is missing}

\def\bplay{\bgroup\global\trno=0
%Version 21/3/90
\def\eplay{\egroup\egroup}
\def\bintermezzo{\noalign\bgroup
\smallskip\noindent}
\def\eintermezzo{\smallskip\egroup}
\tabskipiex plus 1ex minus 1ex
\halign to7\wr\bgroup
\global\advance\trno by 1
\hbox to\wr{\hss\the\trno.\hss} %
\pc\FIP##\hfil&
\pc\SEP##\hfil&
\pc\THP##\hfil&
\pc\FOP##\hfil&
\hfil##\hfil\cr %Template line
\omit\hbox to\wr{\TRICK\hss}&
\omit&\omit&\omit&
\ \NS\ \EW\cr%Headline
}%
} end \bplay

```

```

\def\pc#1#2#3{%
%Function: prints card #2#3 and
%           deletes it from hand #1
%#1 the hand N, E, S, W(uppercase)
%#2 colour s, h, d, or c
%#3 card value A K Q ... 2, or x
%(or your (consistent/language) choice)
%1. Update hand #1#2; e.g. \Ns
\xdef\hnd{\csname #1#2\endcsname}
\strip{#3}{\hnd}%
% end update hand
%2. print card in table
\xdef\colour{\csname #2\endcsname}
\colour\thinspace #3%
% %Needed for immediate * mark
% end print card
}%
} end \pc

```

<sup>8</sup> Of course use of \...lap{\symbol} will not affect the column positioning, but will possibly spoil your print.

```
\def\strip#1#2{%
    Version 3/3/90
    %Function: deletes card value #1
    %           from #2, i.e., \Ns, or ...
\def\wis##1#1##2\wis{%
    %Function: #1 is deleted from argument
    %           in \wis ... \wis and result
    %           is assigned to \hnd;
    %           (last card is replaced by --)
    \global\hnd={##1##2}
    \xdef\pa{##1} \xdef\pb{##2}
    \ifx\pa\empty f\ifx\pb\empty
        \global\hnd={--}% void colour
    \fi\fi
}%
\expandafter\wis\the #2\wis
}%
\end \strip

\def\showgame{%
Shows the play, with
%control sequences Ns, ..., Wc,
%(note use of upper case for player)
%\defs: LFTINF, RGTINF
$$\crdima{\LFTINF}{\RGTINF}%
{\hand{\the\Ns}{\the\Nh}{\the\Nd}%
{\the\Nc}}%
{\hand{\the\Es}{\the\Eh}{\the\Ed}%
{\the\Ec}}%
{\hand{\the\Ss}{\the\Sh}{\the\SD}%
{\the\Sc}}%
{\hand{\the\Ws}{\the\Wh}{\the\Wd}%
{\the\Wc}}%
}%
}%
\end \showgame
```

**Remarks.** Use is made of `\halign`, with a counter for the tricks, and of `\noalign` for the intermezzo. One can also use a third, fourth, etc. symbol, after the colour and card value, in order to denote something special, e.g., !, for a well-played card. I added the reader-friendly feature of printing the cumulative number of tricks gained by each side in extra columns.

One abstraction I consider particularly useful is the notation of `x` for cards which don't matter. (Because of the freedom in representation of card values nothing extra had to be done.)

Another question is what to do when the card is not in the hand? This will yield a TeX error message.

**Flexibility: Endplay Analysis.** The analysis below is due to [15] and shows the elegant use of `\showgame` with (global) control sequences.

Analysis	♠ A8653 ♥ A4 ♦ AJT ♣ A54	7♥, by South
♠ T2	♠ KQ94	
♥ 3	♥ T82	
♦ Q987652	♦ 43	
♣ T86	♣ QJ92	
	♠ J7	
	♥ KQJ9765	
	♦ K	
	♣ K73	

◊2 lead is taken with the K, followed by ♠ to A, ♦A (leader discards a ♠), ♠ trumped, ♥K, ♥ to A, again ♠ trumped, followed by all but one trump. The leader arrived at

Squeeze 1	♠ 8 ♥ - ♦ J ♣ A5	♥5 will squeeze: W (positionally) E (automatically)
♠ -	♠ K	
♥ -	♥ -	
♦ Q	♦ -	
♣ T86	♣ QJ9	
♠ -	♠ -	
♥ 5	♥ 5	
♦ -	♦ -	
♣ K73	♣ K73	

Other squeezes can be envisioned, e.g., (note central figure is suppressed)

Squeeze 2	♠ A8 ♥ - ♦ J ♣ -	W squeezed in ♠/♦
♠ KQ	♠ -	♥ not
♥ -	♦ Q	♦ important
♦ -	♣ -	
♣ -	♠ J7	
	♥ 5	
	♦ -	
	♣ -	

This squeeze works whenever West holds ♠KQ (or 5+♠) and ♦Q, etc.

**Remark.** However interesting other squeeze possibilities — after a trump or ♠ lead — might be, they don't contribute further to 'bridge in print'. The above is meant as an illustration of the use of the macros within the context of a less rigid way of description. Because of the informal way the endplays are arrived at, we edited the hands. General play commands, which will update the hands, are once

again not that difficult to write.<sup>9</sup> For the moment I stopped.

**Input for Endplay Analysis.** The above is obtained via

```
%\local adaptation variables in NESWfig
\def\N{Anton}\def\E{Dick}
\def\S{Frans}\def\W{Rens}
\setbox\NESW\hbox{\NESWfig}
\def\LFTINF{Analysis}
\def\RGTINF{\vtop{\hbox{7\h,}
\hbox{by South}}}
\Ns={A8653}\Es={KQ94}\Ss={J7} \Ws={T2}
\Nh={A4} \Eh={T82} \Sh={KQJ9765}\Wh={3}
\Nd={AJT}\Ed={43} \Sd={K}\Wd={Q987652}
\Nc={A54}\Ec={QJ92}\Sc={K73} \Wc={T86}
%
\showgame
%
%d2 lead is taken with the K, followed by
\s\ to A, \d A (leader discards a \s),
\s\ trumped, \h K, \h\ to A, again
\s\ trumped, followed by all but one
trump. The leader arrived at
\Ns={8} \Es={K} \Ss={--} \Ws={--}
\Nh={--}\Eh={--} \Sh={5} \Wh={--}
\Nd={J} \Ed={--} \Sd={--} \Wd={Q}
\Nc={A5}\Ec={QJ9}\Sc={K73}\Wc={T86}
\def\LFTINF{Squeeze 1}
\def\RGTINF{\vtop{
\hbox{\h5 will squeeze:}
\hbox{W (positionally)}
\hbox{E (automatically)}}}
\showgame
%
Other squeezes can be envisioned, e.g.,
(note central figure is suppressed)
\Ns={A8}\Es={} \Ss={J7} \Ws={KQ}
\Nh={--}\Eh={not}\Sh={5} \Wh={--}
\Nd={J}\Ed={important}\Sd={--}\Wd={Q}
\Nc={--}\Ec={} \Sc={--} \Wc={--}
\def\LFTINF{Squeeze 2}
\def\RGTINF{\vtop{\hbox{W squeezed}
\hbox{in \s/\d}}}
%
{Sublocal mod: empty figure}
```

<sup>9</sup> Informal notation is characterized by incompleteness. In bridge, while discussing the course of a play, it is assumed that the reader knows which player played a card. One could write a general `\strip` command, with a suitable name, which locates the appropriate hand and subsequently strips and prints the card.

```
\setbox\NESW\hbox{}
\showgame
} %end sublocal mod empty figure
%
This squeeze works whenever
West holds \s KQ (or 5$^+$\s) and
\d Q, etc.
} %end local change \NESWfig
```

**Looking back.** I refrained from introducing case insensitivity in the card values, and from automatically counting the gained tricks, which is cumbersome but not too difficult to implement, once a suitable representation for ordering of the cards is chosen.

The above features, as well as more natural input, can best be considered when the macros are targeted for a particular application, e.g., for typesetting (in a specified language) tournament reports, puzzles and answers, or whatever.

Because of the history of `\crdima` and `\hand`, and because I did not much ponder *a priori* about the ‘data structure’, I started with a natural approach. Looking back, I could have started from a 13\*4-matrix, where the rows denote the card values and the columns the colours. The value of an array element represents the status, e.g., the card belongs to either N, E, S, W, or has been played, not to mention ‘penalty’ cards. Updating this structure can be done via the ‘array addressing’ technique given in [9]. `\showgame` (and `\crdima`) as well as `\hand` will become more complicated, however. To be honest, I started in my deal program with 52 numbers for shuffling; these 52 numbers could be generalized into 52 memory locations, suitably addressed.

**Looking ahead.** What about using these macros interactively, e.g., in bridge play programs, or by commentators on TV? Not only to delete a card will be needed but also the reverse, to insert a card, in order to demonstrate variants.<sup>10</sup> Of course, some fancy graphics will be indispensable, like showing real card faces instead of symbols and playing the cards, i.e., letting them *move*. Animation. Multi-media information exchange, how exciting! My case rests.

**Availability of the macros.** This article, with macros included, will be available on `TeX-NL@HEARN`. The previous `LATEX` article is also there. I welcome

<sup>10</sup> Perhaps best implemented via a conditional delete?

copies of any publication using these macros, or derivatives thereof. Comments are appreciated.

### Conclusions

The author claims that bridge publications with respect to card distributions and bidding sequences can be typeset with high quality via L<sup>A</sup>T<sub>E</sub>X, see [12], or via T<sub>E</sub>X and the macros given. Furthermore, it is possible to explain the course of a play in print systematically and unambiguously, where updating of the hands is done automatically when a card is ‘played’, i.e., when within the (`\bplay`, `\eplay`) environment the colour and card value are given, obeying `\halign`’s rules. The display of the course of the play can be interrupted with the intermezzo (sub)environment, for, among others, showing the cards still active in the play via `\showgame`.

Proofreading of deals not generated and typed by computer is error-prone and remains tiresome.

T<sub>E</sub>X programming differs from ‘structured programming’ not in the least

- in terminology — (positional, keyword) parameters vs. arguments, variables vs. registers and control sequences — and
- in its attitude — proving programs vs. knowing what one is doing.

Roughly three columns were needed for the (commented) macros; T<sub>E</sub>X is a powerful tool!

### Acknowledgements

The author is grateful to Bernard Gaulle for his interest in the macros. Johannes Braams, who enlarged the L<sup>A</sup>T<sub>E</sub>X macros into a bridge style file, is kindly acknowledged for emphasizing language flexibility. Victor Eijkhout suggested use of an argument separator for locating a symbol in a string. He also carefully read the manuscript and proposed improvements to my English. Phil Taylor and Amy Hendrickson, whom I met at the Stanford TUG89 conference, and have had T<sub>E</sub>X contacts with since, contributed a lot, not in the least helping me ‘onward and upward’ with the for me unusual way of T<sub>E</sub>X programming. Last but not least I would like to thank the Groningen bridge community for the inspiring discussions and the first class examples.

### References

1. Appelt, W. (1987): Macros with keyword parameters. *TUGboat* 8, no. 2, 182–184.
2. Appelt, W. (1988): Typesetting Chess. *TUGboat* 9, no. 3, 284–287.

3. Bechtolsheim, S. von (1988): Tutorial on `\futurelet`. *TUGboat* 9, no. 3, 276–278.
4. BRIDGE. Monthly of the NBB (Dutch Bridge Union).
5. Coffin, G. S. (1954): Bridge Plays: Four classics. Faber and Faber. London.
6. Crowhurst, E. (1986): ACOL in competition. Pelham. London.
7. Braams, J., V. Eijkhout, N.A.F.M. Poppelier (1990): The development of national L<sup>A</sup>T<sub>E</sub>X styles. *TUGboat* 10, no. 3, 401–406.
8. Glendown, G. (1990): Round boxes for plain T<sub>E</sub>X. *TUGboat* 10, no. 3, 385–386.
9. Greene, A.M. (1989): T<sub>E</sub>Xreation — Playing games with T<sub>E</sub>X’s mind. TUG89 conference, *TUGboat* 10, no. 4, 691–705.
10. Kelder, J., B. van der Velde (1986): Dwangposities tegen één tegenstander. Becht. A’dam. (Dutch). Translated from: Kelsey, H.W. (1985, paperback): Simple squeezes. Gollancz. London.
11. Knuth, D. (1984): The T<sub>E</sub>Xbook. Addison-Wesley.
12. Laan, C.G. van der (1989): Typesetting bridge via L<sup>A</sup>T<sub>E</sub>X. *TUGboat* 10, no. 1, 113–116.
13. Lawrence, M. (1988): The complete book on hand evaluation. Max Hardy. ISBN 0-939460-27-0.
14. Meulenbroek, D. (1989): Nieuwsblad v/h Noorden. December. (Dutch)
15. Paternotte, R. (1989): Dwangneurose. De Enige Goede Bridge Courant. December. (Dutch)
16. Rubinstein, Z. (1990): Printing annotated chess literature in natural notation. *TUGboat* 10, no. 3, 387–389.
17. Vries, H. de (1990): Biedwedstrijd. De Enige Goede Bridge Courant. Maart. (Dutch)

### Appendix. Registers and control sequences used

```
%Card definitions
\def\s{\$spadesuit\$}
\def\h{\$heartsuit\$}
\def\d{\$diamondsuit\$}
\def\c{\$clubsuit\$}
%(Toks register) control sequences
%for hands used by play macros:
%showgame, pc, strip
\let\NT\newtoks
\NT\hnd%Dynamically one of:
\NT\Ns\NT\Es\NT\Ss\NT\Ws
\NT\Nh\NT\Eh\NT\Sh\NT\Wh
\NT\Nd\NT\Ed\NT\Sd
```

```

\NT\Wd %Beware! Already
%in TUGboat.sty in lower case
\NT\Nc\NT\Ec\NT\Sc\NT\Wc

\def\english{
  %In central figure NESW
  \def\N{N}\def\E{E}\def\S{S}\def\W{W}
  %In heading bplay
  \def\NS{NS}\def\EW{EW}
  \def\TRICK{Trick}
  %Definition of hands
  %used by bbid
  \def\FIH{North}\def\SEH{East}
  \def\THH{South}\def\FOH{West}
} % end \english
\english%default

\def\LEADN{\gdef\FIP{N}\gdef\SEP{E}%
  \gdef\THP{S}\gdef\FOP{W}}
\def\LEADE{\gdef\FIP{E}\gdef\SEP{S}%
  \gdef\THP{W}\gdef\FOP{N}}
\def\LEADS{\gdef\FIP{S}\gdef\SEP{W}%
  \gdef\THP{N}\gdef\FOP{E}}
\def\LEADW{\gdef\FIP{W}\gdef\SEP{N}%
  \gdef\THP{E}\gdef\FOP{S}}
%Definition of counters
%used by bplay
\newcount\trno%trick number
%Definition of dimensions

```

◇ Kees van der Laan  
Rekencentrum RUG  
Landleven 1, 9700AV  
Groningen, The Netherlands  
[cgl@rug.nl](mailto:cgl@rug.nl)

## The LATEX Column

Jackie Damrau

In the April 1990 issue, I shared a beginner's type question. This issue I would like to share a macro that was recently needed in my job. I must give proper credit for part of this macro to Norman Richert, University of Houston-Clear Lake.

This example is an excerpt from the *Wolf Cub Scout Book* from the section discussing *Family Discussions about Drug Abuse*.

Should anyone have a macro they would like to share, I urge them to submit the macro no matter how simple it may seem. The macro might be of help to others who are trying to solve a similar problem or help to answer a gray cloud that is hanging over the macro developer.

## Double Item Macro

```
%%%%%%%%%%%%%%%
% Macros for "dblitem.sty"
% Date: March 1990
%
% Jackie Damrau
% SSC Laboratory
% e-mail: damrau@sscvx1
%%%%%%%%%%%%%%%

\topmargin -.5in % Parameters for SSC
\oddsidemargin -.1in % Laboratory print environment
\textheight 8.9in
\textwidth 6.5in

%%%%%%%%%%%%%%
% Special notation needed
%%%%%%%%%%%%%%

\def\header#1#2{\parindent0em{#1 \hfill #2}%
  \vskip.2in \parindent 1.7em}
\def\@oddfoot{\bf \LaTeX\ Column} \hfil \thepage \hfil \today
\def\section{\@startsection{section}{1}{\z@}{+3.5ex plus +1ex
  minus +.2ex}{2.3ex plus .2ex}{\normalsize\rm}}
\def\myth#1{\bf Myth No.\ #1}
\def\fact{\bf Fact}

%%%%%%%%%%%%%%
% Macros for "double items"
% Modified version of \item macro and context on
% p. 355 of TeXBook
%
% Norman J. Richert      March 1990
% Univ. of Houston-Clear Lake
% e-mail: richert@ccl.uh.edu
%%%%%%%%%%%%%%

\newdimen\dblskip           % Register for box w/dblskip width
\dblskip=7.5pc              % Dblskip width of 7.5 pc
\newdimen\firstdbl          % Register for box w/firstdbl width
\firstdbl=6pc               % Firstdbl width of 6 pc
\newdimen\seconddbl          % Register for box w/seconddbl width
\seconddbl=1pc               % Seconddbl width of 1 pc
\def\dblhang{\hangindent\dblskip} % Indent width of dblskip

\def\dbltextindent#1#2{%
  \noindent\hskip\dblskip%   % No indent, horiz. skip of dblskip
  \llap{\hbox to\dblskip{\hbox to\firstdbl{#1 \hfill}}% 
    % left lap with 2 hbox spreads to arg1
  \hbox to \seconddbl{#2\hfil}}}\ignorespaces} % hbox spread to arg2

\def\dblitem{\vskip6pt\dblhang% % Definition of dblitem
  \dbltextindent}
%%%%%%%%%%%%%%
```

## Use of Double Item Macro

Family Discussions

Myths and Facts

Discussing the different myths about drug use is a good way to get children to open up. Listed below are several myths that parents can discuss with their children:

**Myth No. 1** - You won't become addicted to cocaine with casual use.

**Fact** - The two million cocaine addicts will tell you differently. The up-and-down cycle of the cocaine user who always needs more to get a kick is often started with casual use and often continued without the user knowing he or she is becoming addicted.

**Myth No. 2** - One time can't hurt you.

**Fact** - More potent, more available, and more lethal than ever, cocaine, heroin, and a rapidly increasing list of synthetic drugs can threaten the life of even a first-time user. Cocaine, once thought to be less dangerous than other drugs, in 1986 accounted for over 350 deaths. Today's marijuana has three times the amount of THC (the main mind-altering ingredient in marijuana) than marijuana that was available in the 1960s and early 1970s.

Code Used to Produce the Above Example

```
\header{Family Discussions}{Myths and Facts}
```

Discussing the different myths about drug use is a good way to get children to open up. Listed below are several myths that parents can discuss with their children:

```
\dblitem{\myth{1}}{--} You won't become addicted to cocaine with casual use.
```

```
\dblitem{\fact}{--} The two million cocaine addicts will tell you differently. The up-and-down cycle of the cocaine user who always needs more to get a kick is often started with casual use and often continued without the user knowing he or she is becoming addicted.
```

```
\dblitem{\myth{2}}{--} One time can't hurt you.
```

```
\dblitem{\fact}{--} More potent, more available, and more lethal than ever, cocaine, heroin, and a rapidly increasing list of synthetic drugs can threaten the life of even a first-time user. Cocaine, once thought to be less dangerous than other drugs, in 1986 accounted for over 350 deaths. Today's marijuana has three times the amount of THC (the main mind-altering ingredient in marijuana) than marijuana that was available in the 1960s and early 1970s.
```

◊ Jackie Damrau  
 Physics Research Division  
 MS-2002  
 SSC Laboratory  
 2550 Beckleymeade Avenue  
 Suite 260  
 Dallas, TX 75237-3946

(214) 708-6048  
 Bitnet: [damrau@sscvx1](mailto:damrau@sscvx1)  
 Internet: [damrau@sscvx1.ssc.gov](mailto:damrau@sscvx1.ssc.gov)

---

## Making 35 mm Colour Slides with SLiTeX

Ken Yap

When I first read about SLiTeX, and the colour layers feature, I envisioned being able to generate pretty 35 mm slides for talks from SLiTeX input. Some conferences suggest that 35 mm slides be used on account of the large lecture theatres. I was disappointed to discover that slides meant page-size overhead transparencies, and colour meant that one had to obtain special colour materials.

When I saw a colour film recorder, I realized that this was the output device I was looking for. These devices can have resolutions as high as 4096 by 4096 pixels within a 24 mm by 36 mm frame. So I decided to write a dvi-to-35 mm slide converter.

### 1 Overview of SLiTeX

To produce colour slides in SLiTeX, a list of legal colours is declared in the root file thus:

```
\colors{red,green,yellow}
```

Next the slide file is included with the command sequence

```
\colorslides{slides.tex}
```

In the slide file, the list of colours that will be used on a given page is specified thus:

```
\begin{slide}{red,yellow}
```

Any of the legal colours may be used as command sequences or as environment arguments, similar to typeface or size commands. The colour names are meaningful to the author only; SLiTeX has no way of checking what colours will be on the output media.

The reason for the separation of the root file and the slide file becomes clear when one examines the dvi output from a run of SLiTeX: SLiTeX iterates through the colour list with the slide file, so in the example above, first the red pages are generated, then the green pages and finally the yellow pages. Each page appears as many times as colours that it uses. The entities that appear on a layer page are those of the matching colour, or those that are outside the scope of any colour declaration, i.e. pervasive entities. This means that several colour layers have to be combined to yield a slide for each page.

### 2 Implementation

Our film recorder differs from a printer in one major respect: there is no font cache. This device takes an array of pixels and renders that onto the film. The converter has to do the page rasterization.

The backend comprises two programs: dviras and rascombine. The former converts dvi input into a monochrome Sun format *rasterfile*, and the latter combines monochrome rasters, assigning colours, to produce a colour rasterfile.

For dviras, I started with the Nelson Beebe suite of dvi converters because the rasterization code was already written, since Beebe's dvi family includes backends for dot matrix printers. The only hitch encountered was when I realized that the film recorder needed landscape format rasters. A 90° rotation had to be done on portrait pages. I did not want to use a naïve bit-by-bit copying algorithm on a raster of some 2 million bits, so I sought help. Alan Paeth of the University of Waterloo kindly showed me a fast rotation algorithm for 1-bit deep rasters which took under a second for a 2000 by 1000 raster on a Sparcstation I. Problem solved.

The second program, rascombine, assigns colours to each layer, then merges those layers into a single colour raster. The layer colours should be those in the colour list of the root file. A database converts colour names to red/green/blue values, so one can paint layers by name, e.g. vermillion. It is also possible to assign the colours of the background and of entities outside the scope of colour declarations. So "black" lettering can be made purple, if desired!

### 3 Example

```
dviselect 1 < root.dvi > 1.dvi
dviras 1.dvi
rascombine -b gray -o x.ras
    red,green,yellow 1.ras
```

Chris Torek's excellent dviselect program extracts all the layers of page 1 into 1.dvi. Next, the dviras command creates the rasterfile 1.ras with three consecutive rasters. Finally the rascombine command creates a colour rasterfile x.ras with a background of gray and with red, green and yellow text.

### 4 Results

We have just made our first batch of slides with this backend and they look gorgeous. One advantage of using slides for talks is that pictures of experimental results can accompany text slides and one does not have to flitter between the slide projector and the overhead projector.

I hope to try the backend on a colour printer next. This software will be available as part of Nelson Beebe's forthcoming release of his dvi driver family.

## 5 Conclusion

A more elegant and general solution is to merge the pages at the *dvi* level, inserting \specials between entities from different layers. This would have two advantages: the amount of data to be handled is smaller before rasterization, and non-raster devices can be handled by minor modifications to existing drivers. For example, the injected \specials could direct the output device to change colours. This approach would also produce a method for including non-*TEX* graphics in slides; any graphics accepted by the driver and printer could be included, whereas currently, with rasterfiles, only rasters can be merged into the final image, and that with difficulty.

◊ Ken Yap  
 Department of Computer Science,  
 University of Rochester  
 Internet: `ken@cs.rochester.edu`

---

## An Easy Way to Make Slides With *LATEX*

Georg Denk

### Introduction

In the following, a style option for the *article* style will be presented which makes it easy to produce slides. It does not support an overlay structure as *SLiTEX*, but it enables the user to change an *article* to a sequence of slides by simply copying and rearranging. Therefore, it is merely an option for the *article* style and not a new style.

The design of this option fits into the philosophy of *LATEX*: The logical structure of a text is created by the user and not the details of how to put this into a nice output.

The style option described here supports a standard layout which is nevertheless easy to change if necessary. The user is able to think in "normal" dimensions and font sizes as the proper magnification of the slide is done by the output device.

### Some Commands

The style option *eslides* — which stands for *easy slides* and is used to distinguish this option from *SLiTEX* — makes several commands and environments available to the user. These are described in the following. This style option will not conflict with

other style options such as *12pt* or *german*. The option file is not listed here but is available from the author. Send a short request to the e-mail address given below.

### The Magnification of the Slide

The \magnification command gives the global magnification of the slide. As the *dvi* file should be magnified by the same factor, only the quantities 1000, 1095, 1200, 1440, 1728, 2074 and 2488, resp., should be used. A magnification factor of 1440 or 1728 will give good results. Thus one says, e. g.,

```
\magnification{1440}
```

Leslie Lamport has written that *LATEX* should not worry about a magnification of a document. As the output device, however, magnifies everything on the page but the sheet of paper, it is necessary that some of the lengths for the page layout such as \textwidth have to be scaled properly. This is done by the \magnification command. As this procedure is hidden away from the user, he is able to think in "real" dimensions. The previewing will show a correct picture of the slide, usually only somewhat smaller.

If the output device is not able to magnify the slide, it can be done with a photocopying machine, too. The calculation of the various lengths of the page differs as the fixed point of the mapping has changed from the offset point to the middle of the upper edge of the sheet. The *eslides.sty* file contains the necessary commands to handle this.

### Page Layout

The *eslides* style makes the pagestyle *myslide* available to the user:

```
\pagestyle{myslide}
```

Every slide will have a head line and a foot line, separated from the text by a horizontal rule. The head consists of a centered running head which is set with the \markright or \markboth command. The foot line contains a logo, the "name of the conference" and the page number of the slide.

The logo can be anything, e. g., the logo of the university or of the company. In the following examples it is the logo of the Technical University of Munich which is drawn by some *picture* commands. The logo is changed by

```
\renewcommand{\logo}{your logo}
```

As the logo will not change often, probably the best place for the definition will be in the *eslides.sty* file. Perhaps, the logo can be taken from your special *letter.sty*.

Similarly, the conference is set by

```
\renewcommand{\conference}{whatever  
you want}
```

and can be anything, e. g., the name of the conference at which the slides are presented or a \today command.

### The slide Environment

The text of a slide is put between \begin{slide} and \end{slide}. It is vertically centered between the horizontal rules. The user has to take care that the text fits within a single page.

### The remark Environment

The **remark** environment enables the user to create a remark to a slide. This will produce an extra page which contains the note and is numbered with "Remark to ...". The **remark** environment should follow directly the slide to which it belongs:

```
\end{slide}  
\begin{remark}  
...  
\end{remark}
```

### Example

On the following two pages an example for a slide with a remark is given. These pages are produced with a magnification of 1728. The input for the example is shown in the following. The blank titlepage is not presented; it is a trick to produce a running head even on the first page of a document. Normally, the first slide giving the title of the presentation does not need such a running head.

```
\documentstyle[titlepage,eslides]{article}  
% titlepage used, because first page  
% should have a running head  
  
\magnification{1728}  
\renewcommand{\conference}{TUGboat 1990}  
  
\begin{document}  
\pagestyle{myslide}  
\markright{An Easy Way to Make Slides With  
\\LaTeX}  
  
\begin{titlepage}  
\mbox{}  
\end{titlepage}  
% see p. 162 of Lamport's LaTeX book,  
% the first page should have a  
% running head
```

```
\begin{slide}  
\begin{center}  
\bf  
A Short-Cut to Your Slides:  
\end{center}  
\begin{itemize}  
\item take your finished  
{\tt article} file  
\item add the style option  
{\tt eslides}  
\item initialize  
\verb#\magnification#  
and  
\verb#\conference#  
\item put some  
\verb#\begin{slide}#s  
and  
\verb#\end{slide}#s  
around the parts you  
want to present  
\item comment out the rest  
\item run LaTeX  
\end{itemize}  
\end{slide}  
\begin{remark}  
This is a note to myself,  
perhaps reminding me of  
what I wanted to say here,  
e.g.\ that this note is  
stolen from Lamport's  
LaTeX\ book.  
  
\end{remark}  
\end{document}
```

◊ Georg Denk  
Mathematisches Institut  
Technische Universität München  
Arcisstraße 21  
D-8000 München 2  
Bitnet: T1111AA@DMOLRZ01

## An Easy Way to Make Slides With L<sup>A</sup>T<sub>E</sub>X

---

### A Short-Cut to Your Slides:

- take your finished article file
- add the style option `eslides`
- initialize `\magnification` and `\conference`
- put some `\begin{slide}`s and `\end{slide}`s around the parts you want to present
- comment out the rest
- run L<sup>A</sup>T<sub>E</sub>X

## An Easy Way to Make Slides With L<sup>A</sup>T<sub>E</sub>X

---

This is a note to myself, perhaps reminding me of what I wanted to say here, e. g. that this note is stolen from Lamport's L<sup>A</sup>T<sub>E</sub>X book.



---

## A New Implementation of the L<sup>A</sup>T<sub>E</sub>X verbatim and verbatim\* Environments\*

Rainer Schöpf<sup>†</sup>

### Abstract

This style option reimplements the L<sup>A</sup>T<sub>E</sub>X verbatim and verbatim\* environments. In addition it provides a comment environment that skips any commands or text between \begin{comment} and the next \end{comment}. It also contains a redefinition of L<sup>A</sup>T<sub>E</sub>X's \verb command to better detect the omission of the closing delimiter.

### 1 Usage notes

L<sup>A</sup>T<sub>E</sub>X's verbatim and verbatim\* environments have a few features that may give rise to problems. These are:

- Since T<sub>E</sub>X has to read all the text between the \begin{verbatim} and the \end{verbatim} before it can output anything, long verbatim listings may overflow T<sub>E</sub>X's memory.
- Due to the method used to detect the closing \end{verbatim} (i.e. macro parameter delimiting) you cannot leave spaces between the \end token and {verbatim}.

Whereas the last of these points can be considered only a minor nuisance the other one is a real limitation.

This style file contains a reimplementation of the verbatim and verbatim\* environments which overcomes these restrictions. There is, however, one incompatibility between the old and the new implementations of these environments: the old version would treat text on the same line as the \end{verbatim} command as if it were on a line by itself. **This new version will simply ignore it.**<sup>1</sup> It will, however, issue a warning message of the form

LaTeX warning: Characters dropped  
after \end{verbatim}\*!

---

\* This file has version number v1.4a dated 90/04/04. The documentation was last revised on 90/04/04.

† Many thanks to Chris Rowley from The Open University, UK, for looking this over, making a lot of useful suggestions, and discovering bugs. And many thanks to all the beta testers who tried this style file out.

<sup>1</sup> This is the price one has to pay for the removal of the old verbatim environment's size limitations.

This is not a real problem since this text can easily be put on the next line without affecting the output.

This new implementation also solves the second problem mentioned above: it is possible to leave spaces (but *not* end of line) between the \end and the {verbatim} or {verbatim\*}:

```
\begin{verbatim*}
  test
  test
\end{verbatim*}
```

Additionally we introduce a comment environment, with the effect that the text between \begin{comment} and \end{comment} is simply ignored, regardless of what it looks like. At first sight this seems to be quite different from the purpose of verbatim listing, but actually these two concepts turn out to be very similar. Both rely on the fact that the text between \begin{...} and \end{...} is read by T<sub>E</sub>X without interpreting any commands or special characters. The remaining difference between verbatim and comment is only that the text is to be typeset in the former case and to be thrown away in the latter.

\verbatiminput is a command with one argument that inputs a file verbatim, i.e. the command \verbatiminput{xx.yy} has the same effect as

```
\begin{verbatim}
<Contents of the file xx.yy>
\end{verbatim}
```

This command has also a \*-variant that prints spaces as □.

### 2 Interfaces for style file designers

The verbatim environment of L<sup>A</sup>T<sub>E</sub>X version 2.09 does not offer a good interface to programmers. In contrast, this style file provides a simple mechanism to implement similar features, the comment environment provided here being an example of what can be done and how.

#### 2.1 Simple examples

It is now possible to use the verbatim environment to define environments of your own. E.g.,

```
\newenvironment{myverbatim}%
  {\endgraf\noindent MYVERBATIM:}%
  {\endgraf\verbatim}%
  {\end{verbatim}}
```

can be used afterwards like the verbatim environment, i.e.

```
\begin{myverbatim}
  test
  test
\end{myverbatim}
```

Another way to use it is to write

```
\let\foo=\comment
\let\endfoo=\endcomment
```

and from that point on environment `foo` is the same as the `comment` environment, i.e. everything inside its body is ignored.

You may also add special commands after the `\verbatim` macro is invoked, e.g.

```
\newenvironment{myverbatim}%
{\verbatim\myspecialverbatimsetup}%
{\endverbatim}
```

though you may want to learn about the hook `\every@verbatim` at this point. However, there are still a number of restrictions:

1. You must not use `\begin{verbatim}` inside a definition, e.g.

```
\newenvironment{myverbatim}%
{\endgraf\noindent_\MYVERBATIM:}%
\endgraf\begin{verbatim}}%
\end{verbatim}
```

If you try this example, `TEX` will report a “runaway argument” error. More generally, it is not possible to use `\begin{verbatim}...` `\end{verbatim}` or the related environments in the definition of the new environment.

2. You cannot use the `verbatim` environment inside user defined *commands*; e.g.,

```
\newcommand[1]{\verbatimfile}%
\begin{verbatim}%
\input{#1}%
\end{verbatim}}
```

does not work; nor does

```
\newcommand[1]{\verbatimfile}%
{\verbatim\input{#1}\endverbatim}
```

3. The name of the newly defined environment must not contain characters with category code other than 11 (letter) or 12 (other), or this will not work.

## 2.2 The interfaces

Let us start with the simple things. Sometimes it may be necessary to use a special typeface for your `verbatim` text, or perhaps the usual computer modern typewriter shape in a reduced size.

You may select this by redefining the macro `\verbatim@font`. This macro is executed at the beginning of every `verbatim` text to select the font shape. Do not use it for other purposes; if you find yourself abusing this you may want to read about the `\every@verbatim` hook below.

Per default, `\verbatim@font` switches to the typewriter font and disables the ?` and !` ligatures.

There is a hook (i.e. a token register) called `\every@verbatim` whose contents are inserted into `TEX`'s mouth just before every `verbatim` text. Please use the `\addto@hook` macro to add something to this hook. It is used as follows:

```
\addto@hook{name of the hook}
{commands to be added}
```

After all specific setup, like switching of category codes, has been done, the `\verbatim@start` macro is called. This starts the main loop of the scanning mechanism implemented here. Any other environment that wants to make use of this feature should call this macro as its last action.

These are the things that concern the start of a `verbatim` environment. Once this (and other) setup has been done, the code in this style file reads and processes characters from the input stream in the following way:

1. Before it starts to read the first character of an input line the macro `\verbatim@startline` is called.
2. After some characters have been read, the macro `\verbatim@addtoline` is called with these characters as its only argument. This may happen several times per line (when an `\end` command is present on the line in question).
3. When the end of the line is reached, the macro `\verbatim@processline` is called to process the characters that `\verbatim@addtoline` has accumulated.
4. Finally, there is the macro `\verbatim@finish` that is called just before the environment is ended by a call to the `\end` macro.

To make this clear consider the standard `verbatim` environment. In this case the three macros above are defined as follows:

1. `\verbatim@startline` clears the character buffer (a token register).
2. `\verbatim@addtoline` adds its argument to the character buffer.
3. `\verbatim@processline` typesets the characters accumulated in the buffer.

With this it is very simple to implement the `comment` environment: in this case `\verbatim@startline` and `\verbatim@processline` are no-ops whereas `\verbatim@addtoline` discards its argument.

Another possibility is to define a variant of the `verbatim` environment that prints line numbers in the left margin. Assume that this would be done by a counter called `VerbatimLineNo`. Assuming that this counter was initialized properly by the environment, `\verbatim@processline` would be defined in this case as

```
\def\verbatim@processline{%
  \addtocounter{VerbatimLineNo}{1}%
  \llap{\theVerbatimLineNo
  \hspace*{\totalleftmargin}}}
```

As a final nontrivial example we describe the definition of an environment called `\verb+verbatimwrite+`. It writes all text in its body to a file the name of which it is given as an argument. We assume that a stream number called `\verb+verbatim@out+` has already been reserved by means of the `\newwrite` macro.

Let's begin with the definition of the macro `\verb+verbatimwrite+`.

```
\def\verb+verbatimwrite#1{%
```

First we call `\@bsphack` so that this environment does not influence the spacing. Then we open the file and set the category codes of all special characters:

```
\@bsphack
\immediate\openout \verb+verbatim@out #1
```

### 3 The implementation

We use a mechanism similar to the one implemented for the `\comment... \endcomment` macro in *AMS-TEX*: We input one line at a time and check if it contains the `\end{...}` tokens. Then we can decide whether we have reached the end of the verbatim text, or must continue.

As always we begin by identifying the latest version of this file on the VDU and in the transcript file.

```
1 \typeout{Style-Option: 'verbatim'
2   \fileversion \space <\filedate> (RmS)}
3 \typeout{English Documentation
4   \@spaces \@spaces \space <\docdate> (RmS)}
```

#### 3.1 Preliminaries

`\addto@hook` We begin by defining a macro that adds tokens to a hook. The first argument is supposed to be a token register, the second consists of arbitrary *TeX* text.

```
5 \def\addto@hook#1#2{\expandafter{\the#1#2}}
```

`\every@verbatim` The hook (i.e. token register) `\every@verbatim` is initialized to `\emptyset`.

```
6 \newtoks\every@verbatim
7 \every@verbatim={}
```

`\@makeother` `\@makeother` takes as argument a character and changes its category code to 12 (other).

```
8 \def\@makeother#1{\catcode`\#112\relax}
```

`\@vobeyspaces` The macro `\@vobeyspaces` causes spaces in the input to be printed as spaces in the output.

```
9 \begingroup
10 \catcode`\ =\active%
11 \gdef\@vobeyspaces{\catcode`\ \active\let \xobeysp\%}
12 \endgroup
```

```
\let\do\@makeother\dospecials
\catcode`\ ``M\active
```

The default definitions of the macros

```
\verb+at+startline
\verb+at+addtoline
\verb+at+finish
```

are also used in this environment. Only the macro `\verb+verbatim@processline+` has to be changed before `\verb+verbatim@start+` is called:

```
\def\verb+verbatim@processline{%
  \immediate\write\verb+verbatim@out
  {\the\verb+verbatim@line}}%
\verb+verbatim@start{}
```

The definition of `\endverbatimwrite` is very simple: we close the stream and call `\@esphack` to get the spacing right.

```
\def\endverbatimwrite{%
  \immediate\closeout\verb+verbatim@out
  \@esphack}
```

\@xobeysp The macro \@xobeysp produces exactly one space in the output, protected against breaking just before it. (@M is an abbreviation for the number 10000.)

```
13 \def\@xobeysp{\leavevmode\penalty\@M\ }
```

\verbatim@line We use a newly defined token register called \verbatim@line that will be used as the character buffer.

```
14 \newtoks\verbatim@line
```

The following four macros are defined globally in a way suitable for the verbatim and verbatim\* environments.

\verbatim@startline \verbatim@startline initializes processing of a line by emptying the character buffer (\verbatim@line).

```
15 \def\verbatim@startline{\verbatim@line{}}
```

\verbatim@addtoline adds the tokens in its argument to our buffer register \verbatim@line without expanding them.

```
16 \def\verbatim@addtoline#1{%
```

```
17 \verbatim@line\expandafter{\the\verbatim@line#1}}
```

Processing a line inside a verbatim or verbatim\* environment means printing it. Ending the line means that we have to begin a new paragraph. We use \par for this purpose. Note that \par is redefined in \overbatim to force TeX into horizontal mode and to insert an empty box so that empty lines in the input do appear in the output.

```
18 \def\verbatim@processline{\the\verbatim@line\par}
```

\verbatim@finish As a default, \verbatim@finish processes the remaining characters. When this macro is called we are facing the following problem: when the \end{verbatim} command is encountered \verbatim@processline is called to process the characters preceding the command on the same line. If there are none, an empty line would be output if we did not check for this case.

If the line is empty \the\verbatim@line expands to nothing. To test this we use a trick similar to that on p. 376 of the TeXbook, but with \$...\$ instead of the ! tokens. These tokens can never have the same category code as those appearing in the token register \verbatim@line where \$ characters were read with category code 12 (other). Note that \ifcat expands the following tokens so that \the\verbatim@line is replaced by the accumulated characters

```
19 \def\verbatim@finish{\ifcat$\the\verbatim@line$\else
```

```
20 \verbatim@processline\fi}
```

### 3.2 The verbatim and verbatim\* environments

\verbatim@font We start by defining the macro \verbatim@font that is to select the font and to set font-dependent parameters. For the default computer modern typewriter font (cmtt) we have to avoid the ligatures ; and ; (as produced by !' and ?'). We do this by making the backquote ` character active and defining it to insert an explicit kern before the backquote character. While the backquote character is active we cannot use it in a construction like \catcode`<char>=<number>. Instead we use the ASCII code of this character (96).

```
21 \begingroup
```

```
22 \catcode`\`=\active
```

```
23 \gdef\verbatim@font{\tt \catcode96\active \def`{\kern\z@\char96 }}
```

```
24 \endgroup
```

**\@verbatim** The macro `\@verbatim` sets up things properly. First of all, the tokens of the `\every@verbatim` hook are inserted. Then a `trivlist` environment is started and its first `\item` command inserted. Each line of the `verbatim` or `verbatim*` environment will be treated as a separate paragraph.

```
25 \def\@verbatim{\the\every@verbatim
26   \trivlist \item[]%
```

The paragraph parameters are set appropriately: left and right margins, paragraph indentation, the glue to fill the last line and the vertical space between paragraphs. This has to be zero since we do not want to add extra space between lines.

```
27 \leftskip\@totalleftmargin\rightskip\z@
28 \parindent\z@\parfillskip\@flushglue\parskip\z@
```

There's one point to make here: the `list` environment uses `TeX`'s `\parshape` primitive to get a special indentation for the first line of the list. If the list begins with a `verbatim` environment this `\parshape` is still in effect. Therefore we have to reset this internal parameter explicitly. We could do this by assigning 0 to `\parshape`. However, there is a simpler way to achieve this: we simply tell `TeX` to start a new paragraph. As is explained on p. 103 of the `TeXbook`, this resets `\parshape` to zero.

```
29 \@@par
```

We now ensure that `\par` has the correct definition, namely to force `TeX` into horizontal mode and to include an empty box. This is to ensure that empty lines do appear in the output.

```
30 \def\par{\leavevmode\kern-\parindent\par}
```

Now we call `\obeylines` to make the end of line character active,

```
31 \obeylines
```

switch to the font to be used,

```
32 \verbatim@font
```

and change the category code of all special characters to 12 (other).

```
33 \let\do\@makeother \dospecials
```

**\verbatim** Now we define the toplevel macros. `\verbatim` is slightly changed: after setting up things properly it calls `\verbatim@start`.

```
34 \def\verbatim{\@verbatim \frenchspacing\@vobeyspaces\verbatim@start}
```

`\verbatim*` is defined accordingly.

```
35 \namedef{verbatim*}{\@verbatim\verbatim@start}
```

**\endverbatim** To end the `verbatim` and `verbatim*` environments it is only necessary to finish the `trivlist` environment started in `\@verbatim`.

```
36 \let\endverbatim=\endtrivlist
```

```
37 \expandafter\let\csname endverbatim*\endcsname =\endtrivlist
```

### 3.3 The comment environment

**\comment** The `\comment` macro is similar to `\verbatim*`. However, we do not need to switch fonts or set special formatting parameters such as `\parindent` or `\parskip`. We need only set the category code of all special characters to 12 (other) and that of `^M` (the end of line character) to 13 (active). The latter is needed for macro parameter delimiter matching in the internal macros defined below. In contrast to the default definitions used by the `\verbatim` and `\verbatim*` macros, we define `\verbatim@addtoline` to throw away its argument and `\verbatim@processline`, `\verbatim@startline`, and

\verbatim@finish to act as no-ops. Then we call \verbatim@. But the first thing we do is to call \obspack so that this environment has no influence whatsoever upon the spacing.

```

38 \def\comment{\obspack
39           \let\do\makeother\dospecials\catcode`^M\active
40           \let\verbatim@startline\relax
41           \let\verbatim@addtoline\gobble
42           \let\verbatim@processline\relax
43           \let\verbatim@finish\relax
44           \verbatim@}

```

\endcomment is very simple: it only calls \obspack to take care of the spacing. The \end macro closes the group and therefore takes care of restoring everything we changed.

```
45 \let\endcomment=\obspack
```

### 3.4 The main loop

Here comes the tricky part: During the definition of the macros we need to use the special characters \, {, and } not only with their normal category codes, but also with category code 12 (other). We achieve this by the following trick: first we tell TeX that \, {, and } are the lowercase versions of !, [, and ]. Then we replace every occurrence of \, {, and } that should be read with category code 12 by !, [, and ], respectively, and give the whole list of tokens to \lowercase, knowing that category codes are not altered by \lowercase!

But first we have ensure that !, [, and ] themselves have the correct category code! To allow special settings of these codes we hide their setting in the macro \vrb@catcodes. If it is already defined our new definition is skipped.

```

46 \ifundefined{vrb@catcodes}%
47   \def\vrb@catcodes{%
48     \catcode`!=`\lccode`[\lccode`]=`]

```

This allows the use of this code for applications where other category codes are in effect.

We start a group to keep the category code changes local.

```

49 \begingroup
50 \vrb@catcodes
51 \lccode`!=`\lccode`[\lccode`]=`]

```

We also need the end-of-line character ^M, as an active character. If we were to simply write \catcode`^M=\active then we would get an unwanted active end of line character at the end of every line of the following macro definitions. Therefore we use the same trick as above: we write a tilde ~ instead of ^M and pretend that the latter is the lowercase variant of the former. Thus we have to ensure now that the tilde character has category code 13 (active).

```
52 \catcode`~=\active \lccode`~=`^M
```

The use of the \lowercase primitive leads to one problem: the uppercase character 'C' needs to be used in the code below and its case must be preserved. So we add the command:

```
53 \lccode`C=`C
```

Now we start the token list passed to \lowercase.

```
54 \lowercase{%
```

Since this is done in a group all macro definitions are executed globally.

`\verbatim@start` The purpose of `\verbatim@start` is to check whether there are any characters on the same line as the `\begin{verbatim}` and to pretend that they were on a line by themselves. On the other hand, if there are no characters remaining on the current line we shall just find an end of line character. `\verbatim@start` performs its task by first grabbing the following character (its argument). This argument is then compared to an active `^M`, the end of line character.

```
55     \gdef\verbatim@start#1{%
56         \verbatim@startline
57         \if\noexpand#1\noexpand~%
```

If this is true we transfer control to `\verbatim@` to process the next line. We use `\next` as the macro which will continue the work.

```
58     \let\next\verbatim@
```

Otherwise, we define `\next` to expand to a call to `\verbatim@` followed by the character just read so that it is reinserted into the text. This means that those characters remaining on this line are handled as if they formed a line by themselves.

```
59     \else \def\next{\verbatim@#1}\fi
```

Finally we call `\next`.

```
60     \next}%
```

`\verbatim@` The three macros `\verbatim@`, `\verbatim@@`, and `\verbatim@@@` form the “main loop” of the `verbatim` environment. The purpose of `\verbatim@` is to read exactly one line of input. `\verbatim@@` and `\verbatim@@@` work together to find out whether the four characters `\end` (all with category code 12 (other)) occur in that line. If so, `\verbatim@@@` will call `\verbatim@test` to check whether this `\end` is part of `\end{verbatim}` and will terminate the environment if this is the case. Otherwise we continue as if nothing had happened. So let’s have a look at the definition of `\verbatim@`:

```
61     \gdef\verbatim@#1{\verbatim@@#1!end\@nil}%
```

Note that the `!` character will have been replaced by a `\` with category code 12 (other) by the `\lowercase` primitive governing this code before the definition of this macro actually takes place. That means that it takes the line, puts `\end` (four character tokens) and `\@nil` (one control sequence token) as a delimiter behind it, and then calls `\verbatim@@`.

`\verbatim@@` `\verbatim@@` takes everything up to the next occurrence of the four characters `\end` as its argument.

```
62     \gdef\verbatim@@#1!end{%
```

That means: if they do not occur in the original line, then argument `#1` is the whole input line, and `\@nil` is the next token to be processed. However, if the four characters `\end` are part of the original line, then `#1` consists of the characters in front of `\end`, and the next token is the following character (always remember that the line was lengthened by five tokens). Whatever `#1` may be, it is verbatim text, so `#1` is added to the line currently built.

```
63     \verbatim@addtoline{#1}%
```

The next token in the input stream is of special interest to us. Therefore `\futurelet` defines `\next` to be equal to it before calling `\verbatim@@@`.

```
64     \futurelet\next\verbatim@@@}%
```

`\verbatim@@@` `\verbatim@@@` will now read the rest of the tokens on the current line, up to the final `\@nil` token.

```
65     \gdef\verbatim@@@#1\@nil{%
```

If the first of the above two cases occurred, i.e. no \end characters were on that line, #1 is empty and \next is equal to \nil. This is easily checked.

```
66      \ifx\next\@nil
```

If so, this was a simple line. We finish it by processing the line we accumulated so far. Then we prepare to read the next line.

```
67      \verbatim@processline
68      \verbatim@startline
69      \let\next\verbatim@
```

Otherwise we have to check what follows these \end tokens.

```
70      \else
```

Before we continue, it's a good idea to stop for a moment and remember where we are: We have just read the four character tokens \end and must now check whether the name of the environment (surrounded by braces) follows. To this end we define a macro called \tempa that reads exactly one character and decides what to do next. This macro should do the following: skip spaces until it encounters either a left brace or the end of the line. But it is important to remember which characters are skipped. The \end<optional spaces>{ characters may be part of the verbatim text, i.e. these characters must be printed.

Assume for example that the current line contains

```
.....\end{AVeryLongEnvironmentName}
```

As we shall soon see, the scanning mechanism implemented here will not find out that this is text to be printed until it has read the right brace. Therefore we need a way to accumulate the characters read so that we can reinsert them if necessary. The token register \temptokena is used for this purpose.

Before we do this we have to get rid of the superfluous \end tokens at the end of the line. To this end we define a temporary macro whose argument is delimited by \end\@nil (four character tokens and one control sequence token) and use it on the rest of the line, after appending a \nil token to it. This token can never appear in #1. We use the following definition of \tempa to store the rest of the line (after the first \end) in token register \toks@ which we shall use again in a moment.

```
71      \def\tempa##1\end\@nil{\toks@{\#1}%
72      \tempa#1\@nil
```

We mentioned already that we use token register \temptokena to remember the characters we skip, in case we need them again. We initialize this with the \end we have thrown away in the call to \tempa.

```
73      \temptokena{!end}%
```

We shall now call \verbatim@test to process the characters remaining on the current line. But wait a moment: we cannot simply call this macro since we have already read the whole line. We stored its characters in token register \toks@. Therefore we use the following \edef to insert them again after the \verbatim@test token. A ^M character is appended to denote the end of the line.

```
74      \edef\next{\noexpand\verbatim@test\the\toks@\noexpand^M}%
```

That's almost all, but we still have to now call \next to do the work.

```
75      \fi \next}%
```

\verbatim@test We define \verbatim@test to investigate every token in turn.

```
76  \gdef\verbatim@test#1{%
```

First of all we set `\next` equal to `\verbatim@test` in case this macro must call itself recursively in order to skip spaces.

```
77      \let\next\verbatim@test
```

We have to distinguish four cases:

1. The next token is a `^M`, i.e. we reached the end of the line. That means that nothing special was found. Note that we use `\if` for the following comparisons so that the category code of the characters is irrelevant.

```
78      \if\noexpand#1\noexpand~%
```

We add the characters accumulated in token register `\@temptokena` to the current line. Since `\verbatim@addtoline` does not expand its argument, we have to do the expansion at this point. Then we `\let \next` equal to `\verbatim@` to prepare to read the next line.

```
79      \expandafter\verbatim@addtoline
80          \expandafter{\the\@temptokena}%
81      \verbatim@processline
82      \verbatim@startline
83      \let\next\verbatim@
```

2. A space character follows. This is allowed, so we add it to `\@temptokena` and continue.

```
84      \else \if\noexpand#1
85          \@temptokena\expandafter{\the\@temptokena#1}%
```

3. An open brace follows. This is the most interesting case. We must now collect characters until we read the closing brace and check whether they form the environment name. This will be done by `\verbatim@testend`, so here we let `\next` equal this macro. Again we will process the rest of the line, character by character. The characters forming the name of the environment will be accumulated in `\@tempc`. We initialize this macro to expand to nothing.

```
86      \else \if\noexpand#1\noexpand[%
87          \let\@tempc\@empty
88          \let\next\verbatim@testend
```

Note that the `[` character will be a `{` when this macro is defined.

4. Any other character means that the `\end` was part of the verbatim text. Add the characters to the current line and prepare to call `\verbatim@` to process the rest of the line.

```
89      \else
90          \expandafter\verbatim@addtoline
91              \expandafter{\the\@temptokena}%
92          \def\next{\verbatim@#1}%
93          \fi\fi\fi
```

The last thing this macro does is to call `\next` to continue processing.

```
94      \next}%
```

`\verbatim@testend` `\verbatim@testend` is called when `\end<optional spaces>{` was seen. Its task is to scan everything up to the next `}` and to call `\verbatim@@testend`. If no `}` is found it must reinser the characters it read and return to `\verbatim@`. The following definition is similar to that of `\verbatim@test`: it takes the next character and decides what to do.

```
95      \gdef\verbatim@testend#1{%
```

Again, we have four cases:

1. ^^M: As no } is found in the current line, add the characters to the buffer. To avoid a complicated construction for expanding \Ctempokena and \Ctempc we do it in two steps. Then we continue with \verbatim@ to process the next line.

```

96      \if\noexpand#1\noexpand%
97      \expandafter\verbatim@addtoline
98      \expandafter{\the\Ctempokena[]}%
99      \expandafter\verbatim@addtoline
100     \expandafter{\Ctempc}%
101     \verbatim@processline
102     \verbatim@startline
103     \let\next\verbatim@

```

2. }: Call \verbatim@@testend to check if this is the right environment name.

```

104    \else\if\noexpand#1\noexpand]%
105    \let\next\verbatim@@testend

```

3. \: This character must not occur in the name of an environment. Thus we stop collecting characters. In principle, the same argument would apply to other characters as well, e.g., {. However, \ is a special case, since it may be the first character of \end. This means that we have to look again for \end{*environment name*}. Note that we prefixed the ! by a \noexpand primitive, to protect ourselves against it being an active character.

```

106    \else\if\noexpand#1\noexpand!%
107    \expandafter\verbatim@addtoline
108    \expandafter{\the\Ctempokena[]}%
109    \expandafter\verbatim@addtoline
110    \expandafter{\Ctempc}%
111    \def\next{\verbatim@!}%

```

4. Any other character: collect it and continue. We cannot use \edef to define \Ctempc since its replacement text might contain active character tokens.

```

112    \else \expandafter\def\expandafter\@tempc\expandafter
113    {\@tempc#1}\fi\fi\fi

```

As before, the macro ends by calling itself, to process the next character if appropriate.

```
114    \next}%

```

**\verbatim@@testend** Unlike the previous macros \verbatim@@testend is simple: it has only to check if the \end{...} matches the corresponding \begin{...}.

```
115    \gdef\verbatim@@testend{%

```

We use \next again to define the things that are to be done. Remember that the name of the current environment is held in \Currenvir, the characters accumulated by \verbatim@testend are in \Ctempc. So we simply compare these and prepare to execute \end{*current environment*} macro if they match. Before we do this we call \verbatim@finish to process the last line. We define \next via \edef so that \Currenvir is replaced by its expansion. Therefore we need \noexpand to inhibit the expansion of \end at this point.

```

116    \ifx\@tempc\Currenvir
117    \verbatim@finish
118    \edef\next{\noexpand\end{\Currenvir}}%

```

Without this trick the \end command would not be able to correctly check whether its argument matches the name of the current environment and you'd get an interesting LATEX error message such as:

```
! \begin{verbatim*} ended by \end{verbatim*}.
```

But what do we do with the rest of the characters, those that remain on that line? We call `\verbatim@rescan` to take care of that. Its first argument is the name of the environment just ended, in case we need it again. `\verbatim@rescan` takes the list of characters to be reprocessed as its second argument. (This token list was inserted after the current macro by `\verb@QQ`.) Since we are still in an `\edef` we protect it by means of `\noexpand`.

```
119          \noexpand\verbatim@rescan{\@currenvir}}%
```

If the names do not match, we reinsert everything read up to now and prepare to call `\verbatim@` to process the rest of the line.

```
120      \else
121          \expandafter\verbatim@addtoline
122          \expandafter{\the\@temptokena[]}
123          \expandafter\verbatim@addtoline
124          \expandafter{\@tempc[]}
125          \let\next\verbatim@
126      \fi
```

Finally we call `\next`.

```
127          \next}%
```

#### `\verbatim@rescan`

In principle `\verbatim@rescan` could be used to analyse the characters remaining after the `\end{...}` command and pretend that these were read “properly”, assuming “standard” category codes are in force.<sup>2</sup> But this is not always possible (when there are unmatched curly braces in the rest of the line). Besides, we think that this is not worth the effort: After a `\verbatim` or `\verbatim*` environment a new line in the output is begun anyway, and an `\end{comment}` can easily be put on a line by itself. So there is no reason why there should be any text here. For the benefit of the user who did put something there (a comment, perhaps) we simply issue a warning and drop them. The method of testing is explained in Appendix D, p. 376 of the *TeXbook*. We use `^^M` instead of the ! character used there since this is a character that cannot appear in #1. The two `\noexpand` primitives are necessary to avoid expansion of active characters and macros.

One extra subtlety should be noted here: remember that the token list we are currently building will first be processed by the `\lowercase` primitive before *TeX* carries out the definitions. This means that the ‘C’ character in the argument to the `\@warning` macro must be protected against being changed to ‘c’. That’s the reason why we added the `\lccode`'C='`C` assignment above. We can now finish the argument to `\lowercase` as well as the group in which the category codes were changed.

```
128  \gdef\verbatim@rescan#1#2{\if\noexpand^\noexpand#2^{\else
129      \@warning{Characters dropped after '\string\end{#1}'}\fi}}
130 \endgroup
```

#### 3.5 The `\verbatiminput` command

`\verbatiminput` first starts a group to keep font and category changes local.

```
131 \def\verbatiminput{\begingroup
```

The right sequence of actions is crucial here. First we must check if a star follows. Then we must read the argument (the file name). Finally we must set up everything to read the contents of the file `verbatim`. Therefore we must not start by calling `\@verbatim` to change font and the category code of characters. Instead we call one

---

<sup>2</sup> Remember that they were all read with category codes 11 (letter) and 12 (other) so that control sequences are not recognized as such.

of the macros `\sverbatim@input` or `\verbatim@input`, depending on whether a star follows.

```
132  \@ifstar\sverbatim@input\verbatim@input}
```

`\sverbatim@input` `\sverbatim@input` reads the file name argument and sets up everything as in the `\verbatim` macro. Then it reads in the file, finishes off the `trivlist` environment started by `\@verbatim` and closes the group opened in `\verbatininput`. This restores everything to its normal settings.

```
133 \def\sverbatim@input#1{\@verbatim
134   \@input{#1}\endtrivlist\endgroup\@doendpe}
```

`\verbatim@input` `\verbatim@input` is nearly the same; it additionally calls `\frenchspacing` and `\vobeyspaces` (as in `\verbatim` and `\verb`).

```
135 \def\verbatim@input#1{\@verbatim
136   \frenchspacing \vobeyspaces
137   \@input{#1}\endtrivlist\endgroup\@doendpe}
```

### 3.6 Redefinition of the `\verb` command.

The implementation here has the following advantage over that in the original L<sup>A</sup>T<sub>E</sub>X: it will not accept that the end of the input line is reached before the verbatim text has ended. Instead, it will end the verbatim text and generate an error message.

`\verb` We need special category codes during the definition: the end of line character (`^M`) must be an active character. We do this in the same way as above:

```
138 \begingroup
139   \lccode`~`\^M
140   \lowercase{%
141     \gdef\verb{\begingroup
```

We use here `\verb@font` rather than switching directly to `\tt`.

```
142   \verb@font
```

Now we make the end of line character active and define it to restore everything back to normal and to signal an error.

```
143   \def~{\endgroup\@latexerr{\string\verb\space command ended by
144   end of line.}\@ehc}%
```

The rest is copied from `latex.tex` where we have replaced one macro (`\@verb`) by its expansion.

```
145   \let\do\@makeother \dospecials
146   \ifstar\@sverb{\vobeyspaces \frenchspacing \@sverb}}}
147 \endgroup
```

`\@sverb` `\@sverb` gains control when we are ready to look for the delimiting character. It reads it and defines this character to be equivalent to the `\endgroup` primitive. I.e. it will restore everything to normal when it occurs for the second time. But this is not enough: if the first character of `\verb`'s argument is a space and if a line break occurs at this point the space will still disappear. To avoid this we include an empty `\hbox{}` at the beginning.

```
148 \def\@sverb#1{%
149   \catcode`#1\active
150   \lccode`~`#1%
151   \lowercase{\let~\endgroup}%
152   \leavevmode\null}
```

## Index

The italic numbers denote the lines where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

Symbols	V
\@makeother .....	<u>8</u>
\@sverb .....	<u>148</u>
\@verbatim .....	<u>25</u>
\@vobeyspaces .....	<u>9</u>
\@xobeysp .....	<u>13</u>
<b>A</b>	
\addto@hook .....	<u>5</u>
<b>C</b>	
\comment .....	<u>38</u>
<b>E</b>	
\endcomment .....	<u>38</u>
\endverbatim .....	<u>36</u>
\endverbatim* .....	<u>36</u>
\every@verbatim .....	<u>6</u>
<b>S</b>	
\sverbatim@input .....	<u>133</u>
<b>V</b>	
\verb .....	<u>138</u>
\verbatim .....	<u>34</u>
\verbatim* .....	<u>34</u>
\verbatim@ .....	<u>61</u>
\verbatim@C .....	<u>62</u>
\verbatim@CC .....	<u>65</u>
\verbatim@Ctestend .....	<u>115</u>
\verbatim@addtoline .....	<u>15</u>
\verbatim@finish .....	<u>19</u>
\verbatim@font .....	<u>21</u>
\verbatim@input .....	<u>135</u>
\verbatim@line .....	<u>14</u>
\verbatim@processline .....	<u>15</u>
\verbatim@rescan .....	<u>128</u>
\verbatim@start .....	<u>55</u>
\verbatim@startline .....	<u>15</u>
\verbatim@test .....	<u>76</u>
\verbatim@testend .....	<u>95</u>
\verbatim@input .....	<u>131</u>

◊ Rainer Schöpf  
 Institut für Theoretische Physik  
 der Universität Heidelberg  
 Philosophenweg 16  
 D-6900 Heidelberg  
 Federal Republic of Germany

Editor's note: We regret to say that a large piece of Frank and Rainer's article below was inadvertently deleted from the file used to publish the article in *TUGboat* 11, no. 1. Under the section-numbering presented here, the previously missing portion extends from the end of section 1 to subsection 3.2. We feel it is best to reprint the article entirely, and apologize for any confusion or misunderstanding this has caused our readership.

*Reprint (with corrections):*  
**The New Font Family Selection —  
User Interface to Standard L<sup>A</sup>T<sub>E</sub>X**

Frank Mittelbach  
Rainer Schöpf

**Contents**

<b>1 General remarks</b>	<b>297</b>
<b>2 Choosing a new text font</b>	<b>298</b>
2.1 Choosing a new <i>family</i>	
2.2 Choosing a new <i>series</i>	
2.3 Choosing a new <i>shape</i>	
2.4 Choosing a new <i>size</i>	
2.5 Doing it by hand	
2.6 Changing the meaning of \rm, \sf etc.	
<b>3 Fonts for Math</b>	<b>300</b>
3.1 Simple formulas	
3.2 Special math alphabets	
<b>4 Processing older documents</b>	<b>302</b>
<b>5 Setting up a new format</b>	<b>302</b>
5.1 Preparations	
5.1.1 Preloading Fonts	
5.1.2 Making more fonts available	
5.2 Running IniTeX	
<b>6 Remarks on the development of this interface</b>	<b>304</b>
<b>7 Acknowledgements</b>	<b>304</b>
<b>8 List of distributed files</b>	<b>304</b>

**Abstract**

In this article we describe the use of the new font selection scheme in the standard L<sup>A</sup>T<sub>E</sub>X environment. The main characteristics are:

- The possibility to change family, series, shape and sizes independently of one another.
- The existence of a style file to process older documents without any changes to their layout and their input files.
- A macro setup which is consistent with existing standard document styles.<sup>1</sup>

It is planned to incorporate this font selection scheme into L<sup>A</sup>T<sub>E</sub>X version 2.10.

**1 General remarks**

In *TUGboat* 10, no. 2 we presented a new scheme to select fonts in TeX macro packages. This article describes the use of this new scheme in the L<sup>A</sup>T<sub>E</sub>X environment. The technical parts of the interface (which are of some interest to readers who plan to use our scheme with other fonts or with other macro packages) will be published in a separate article.

The necessary macros are distributed by the AMS together with the amstex.sty option which was announced in *TUGboat* 10, no. 3. The availability of the new font selection scheme at the usual servers will be announced separately in TeXhax, etc. Please refrain from asking for personal distribution.

To get a better understanding of this L<sup>A</sup>T<sub>E</sub>X interface, some words on the organisation of font families are in order. Readers of our article about the basic macros will notice that our understanding of these matters increased while working on this interface and the amstex.sty project; in some regards we have changed our point of view rather drastically. Surprisingly, only a few internal details within the basic macros needed adjustment; it seems that even without the real understanding, we instinctively got most of the things right when we designed them. (But probably we are still ignorant of the underlying concepts.)

In his book about "Methods of Book Design" Hugh Williamson writes [1]

[...] To the printer, an alphabet is a set of twenty-six letters of a certain design and body, together with a few additional combinations of letters. A *fount* is usually made up of a set of alphabets of one size and based on one design. It may consist of one alphabet only, if no more alphabets exist in that design and size. Usually however a text fount will comprise five alphabets — roman and

<sup>1</sup> However, small changes in the document styles would make font changes a bit faster.

italic upper and lower-case, and small capitals. [...] A *series* is a set of founts closely related to each other in design, and usually very similar to each other, but graded in size. If only one alphabet has been made in a certain design, that alphabet alone may be a series. A *family* is a group of series compatible for composition, but loosely related in design. A family may include excerpts from more than one series.

Since TeX's physical fonts (which is the American word for fount) all contain exactly two alphabets, namely the upper and lower-case alphabets of a certain design, we will use the word font for physical TeX fonts, and fount for bundles of TeX fonts consisting for example of roman (upright or normal), italic and small capitals shapes.

The above quotation gives a good clue how to organize fonts in our font selection scheme. Hence we use the `\shape` command from the basic macros to distinguish between normal (n), italic (it), small caps (sc), sloped or slanted (sl) and upright italic (u) typefaces within one fount. Founts of different sizes form a series, so we use `\size` to access these. We think that the weight and the width of a series are good candidates to distinguish between individual series, therefore we combine them in the `\series` command. Again we use one and two letter abbreviations as shown in table 1. One or more of these series form a family which is accessed via the `\family` command.

To give some practical example, we arranged the most important families of the Computer Modern fonts according to this classification in table 2. Please note that some families like 'computer modern funny roman' (cmff) or 'computer modern sans serif quotation' (cmssq) are unclassified. These special purpose fonts are not accessible in the standard distribution of the new font selection scheme, although they could be added easily in a style file.

Given this overview about the classification of fonts it should be clear how to select a specific font with the primitive commands `\family`, `\series`, `\shape`, `\size`, and `\selectfont`. As described in [5], the `\size` macro takes two arguments: the size in printer's points as a numeral (i.e. without the dimension) and the corresponding `\baselineskip` value. `\selectfont` finally selects the font using values of the surrounding environment if some of the commands are missing. E.g., the sequence

```
\family{ccr}\series{c}\shape{sl}%
\size{9}{11pt}\selectfont
```

will explicitly load the font mentioned above, provided the necessary font shapes are known to the system.<sup>2</sup>

However, in the normal case, the L<sup>A</sup>T<sub>E</sub>X user can safely rely on the standard L<sup>A</sup>T<sub>E</sub>X font selection commands defined in terms of these primitive commands. These standard commands are discussed in the next sections.

## 2 Choosing a new text font

In standard L<sup>A</sup>T<sub>E</sub>X, different fonts (of the same size) are selected by commands like `\rm`, `\bf`, `\it`, etc. These commands, however, select a specific font regardless of surrounding conditions. E.g., if you write `\sf\bf` you don't get the 'bold extended' *series* of the 'sans serif' *family* (i.e. cmssbx), instead you get the 'bold extended' *series* of the 'roman' *family* (i.e. cmbx). In our implementation this will be different.

Commands like `\bf` (or `\sf`) are now implemented to switch to the wanted *series* (or *family*, respectively), but to leave the other font characteristics untouched. However, this concept has one drawback in the current L<sup>A</sup>T<sub>E</sub>X version: commands like `\footnote` might switch to a smaller size but will inherit other characteristics for the font to use from the environment where they are used. E.g., a footnote appearing in the scope of a theorem environment will erroneously be typeset in italic shape. This problem will vanish in L<sup>A</sup>T<sub>E</sub>X version 2.10. Until then all font characteristics in such special circumstances must be reset by hand using the commands given below. However, this is not necessary if one uses the 'oldlf' style option described in sections 4 and 5. This option defines the font selection commands to behave in the same way as they do now in L<sup>A</sup>T<sub>E</sub>X 2.09.

### 2.1 Choosing a new *family*

To switch to another *family* one may use `\rm`, `\sf` or `\tt` denoting the 'cmroman', 'cmsansserif' or 'cmtypewriter' *family*, respectively. The new font is selected without changing the current *series*, *shape*,

---

<sup>2</sup> Among the AMS distribution an example style option 'concrete.sty' is provided which makes the Concrete roman as well as the Euler math fonts available. These fonts were used to typeset [4] and this article.

Weight Class		Width Class		
Ultralight	ul	Ultracondensed	50%	uc
Extralight	el	Extracondensed	62.5%	ec
Light	l	Condensed	75%	c
Semilight	sl	Semicondensed	87.5%	sc
Medium (normal)	m	Medium	100%	m
Semibold	sb	Semieexpanded	112.5%	sx
Bold	b	Expanded	125%	x
Extrabold	eb	Extraexpanded	150%	ex
Ultrabold	ub	Ultraexpanded	200%	ux

Table 1: Weight and width classification for fonts. The percent values are derived from [2]. To combine the abbreviations in the \series command, weight is used first and any instance of medium (m) is dropped except when weight and width are both medium. In this case one single m is used. So bold expanded would be bx whereas medium expanded would be x.

### Computer Modern families

family	series	shape(s)	Example of external names
<i>Computer modern roman</i>			
cmr	m	n, it, sl, sc, u	cmr10, cmti10, cmsl10, cmcsc10, cmu10
cmr	bx	n, it, sl	cmbx10, cmbxti, cmbxsl
cmr	b	n	cmb10
<i>Computer modern sans serif</i>			
cmss	m	n, sl	cmss10, cmssi10
cmss	bx	n	cmssbx10
cmss	sbc	n	cmssdc10
<i>Computer modern typewriter</i>			
cmtt	m	n, it, sl, sc	cmtt10, cmiti10, cmsltt, cmtcsc10
<i>Computer modern fibonacci</i>			
cmfi	m	n	cmfib8

Table 2: Classification of the Computer modern fonts. You will notice that not all possible combinations of family, series and shape are available. E.g. there is no small capitals shape in the medium series of the computer modern sans serif. However, Philip Taylor announced recently that he has filled some of the holes. It might be a good idea to include such additional parameter files for METAFONT into the general distributions.

and *size*. E.g., if the current font is cmbx10 (that is *family* ‘computer modern roman’, *series* ‘bold extended’, *shape* ‘normal’, and *size* ‘10pt’) then *\sf* will change to cmssbx10 (that is *family* ‘computer modern sans serif’, *series* ‘bold extended’, *shape* ‘normal’, and *size* ‘10pt’). Using, e.g., *\tt* afterward will produce a warning and switch to cmtt10 because the ‘computer modern typewriter’ *family* does not contain a ‘bold extended’ *series*; therefore the default (‘medium’) *series* is tried.

## 2.2 Choosing a new *series*

To switch between ‘medium’ and ‘bold extended’ *series* the commands *\mediumseries* and *\bf* are provided.

## 2.3 Choosing a new *shape*

Analogously the commands *\sl*, *\it*, and *\sc* are used to switch to the *shapes* ‘sloped’, ‘italic’ and ‘smallcaps’, this time leaving *family*, *series*, and *size* alone.

In addition, we introduce the *\normalshape* command, in case one wants to switch back to the ‘normal’ *shape*. If font changes are done only inside of groups this command is necessary only to reset a shape in a footnote or a similar context to avoid the problem mentioned above.

## 2.4 Choosing a new *size*

To change to another *size* the standard L<sup>A</sup>T<sub>E</sub>X commands

```
\tiny   \scriptsize  \footnotesize
\small  \normalsize  \large
\Large  \LARGE      \huge and \Huge
```

may be used. These commands also set the parameter *\baselineskip* and the *\strutbox* as well as the script and scriptscript *sizes* for the new text *size*. But once again they will not change other font characteristics. So, for example, it doesn’t matter whether one writes *\large\tt* or *\tt\large*, the same font will be selected.

## 2.5 Doing it by hand

As mentioned before, primitive font selection commands like *\family*, *\series*, *\shape*, *\size*, and *\selectfont* are also available to carry out the change. E.g.,

```
\shape{n}\family{cmss}\selectfont
```

will switch to the ‘cmsansserif’ family with ‘normal’ shape.<sup>3</sup> This article was set in concrete roman type by simply writing *\family{ccr}\selectfont* following the *\begin{document}* command.<sup>4</sup> Of course, the *\documentstyle* command also specifies a style option (‘concrete’) which sets up the internal tables for these fonts.

## 2.6 Changing the meaning of *\rm*, *\sf* etc.

To make it easy to typeset documents with other font families (like Times Roman, Optima etc.) we maintain seven additional macros

```
\rmdefault \sfdefault \ttdefault \bfdefault
\itdefault \scdefault \sldefault
```

denoting the family chosen by *\rm*, *\sf*, *\tt* or the series chosen by *\bf* or the shape for *\it*, *\sc* and *\sl*, respectively.<sup>5</sup> If, for example, a document should be typeset in sans serif one could add in the preamble (between *\documentstyle* and *\begin{document}*) the following redefinitions:

```
\renewcommand{\rmdefault}{cmss}
\renewcommand{\itdefault}{sl}
```

The first line means that whenever *\rm* is called the family cmss (i.e. computer modern sans serif) is chosen and the second line redefines *\it* to switch to the slanted shape since this family hasn’t got an *italic* shape. Another possible use is to say

```
\renewcommand{\bfdefault}{b}
```

This will redefine *\bf* to select the bold instead of the bold extended series which is the current default. However these commands are probably ignored by document styles for journals which decide to use their own font families in the final print.

## 3 Fonts for Math

The selection of a specific typeface in a math formula should not depend on the surrounding environment. Characters in math normally denote special things which should stay fixed even if the surrounding text is set in another *shape* or *series*. Therefore the strategy for selecting math fonts is somewhat different.

<sup>3</sup> For full details of the usage of these primitive commands see the article about the basic macros [5].

<sup>4</sup> Actually we also said *\size{10}{13pt}* to establish a larger *\baselineskip*.

<sup>5</sup> This was suggested by Sebastian Rahtz who was one of the first users of our prototype version.

### 3.1 Simple formulas

Normal letters and standard symbols are typeset simply by using the letters directly or using a command that denotes the wanted symbol. So  $\sum A_i$  will produce  $\sum A_i$ . The typeface chosen will depend on the current *(math version)*. You can switch between *(math versions)* outside of math mode,<sup>6</sup> thereby changing the overall layout of the following formulas.

LATEX knows about two versions called ‘normal’ and ‘bold’. As the name indicates,  $\mathit{\mathbf{mathversion{normal}}}$  is the default. In contrast, the bold version will produce bolder letters and symbols. This might be suitable in certain situations, but recall that changing the version means changing the appearance (and perhaps the meaning) of the whole formula. For historical reasons LATEX maintains two abbreviations to switch to its math versions:  $\boldsymbol{\mathit{\mathbf{boldmath}}}$  and  $\mathit{\mathbf{unboldmath}}$ .

Other versions could be provided in special style options. For example the ‘concrete’ option mentioned before sets up a version called ‘euler’ to typeset formulas in the same way as it was done in [4].

### 3.2 Special math alphabets

But simple formulas with one alphabet and a huge number of symbols are not sufficient for mathematicians to expose their thoughts properly. They tend to use every available typeface to denote special things.

To cope with this need for special alphabets in formulas,’ we introduce the concept of *(math alphabet identifiers)*. These constructs are special commands which switch to a specific typeface. They might correspond to different typefaces in different math versions but within one version they always select the same typeface regardless of surrounding conditions.

A *(math alphabet identifier)* can be defined according to the users’ needs but standard LATEX already has a few of them built in. They are described in table 3.

When using such an *(alphabet identifier)* two syntax variants are available: one can understand a command like  $\mathcal{A}$  as a switch to a different font, i.e. using a syntax  $\{\mathcal{A} \dots\}$  as the old LATEX does, but we prefer to view the *(math alphabet*

*identifier*) as a command with one argument, i.e. to use a syntax of the form  $\{\mathcal{A}\}$ . To select the first alternative a style option ‘nomargid’ is provided. This option is automatically selected if the ‘oldlfont’ option is used since this option is supposed to produce identical results for older documents.

New *(math alphabet identifiers)* are defined in two steps. First the identifier is made known to the system with the  $\mathit{\mathbf{newmathalphabet}}$  command. Then specific typefaces in some or all *(math versions)* are assigned by means of the  $\mathit{\mathbf{addtoversion}}$  command.

Let us discuss this process in detail. Suppose that you want to make a sans serif typeface available as a math alphabet. First we choose a new command name (e.g.  $\mathit{\mathbf{sfmath}}$ ) and tell LATEX about it with the line

```
\mathit{\mathbf{newmathalphabet{\sfmath}}}
```

Then we consult table 2 to find suitable fonts to assign to this alphabet identifier. As you find out, the computer modern sans serif family consists of three series, a medium, semi bold condensed and a bold extended one. The medium and the bold extended series both contain a normal shape typeface. So we add the line:

```
\mathit{\mathbf{addtoversion{normal}{\sfmath}{cmss}{m}{n}}}
\mathit{\mathbf{addtoversion{bold}{\sfmath}{cmss}{bx}{n}}}
```

Now our alphabet identifier is ready for use in these two versions. We demonstrate this with the formula

$$\sum A_i = \tan \alpha$$

which was produced by

```
\mathit{\mathbf{mathversion{normal}}}
\mathit{\mathbf{[ \sum \sfmath{A}_i = \tan \alpha ]}}
```

Note that we first switched back to the normal version. This was necessary since this article is typeset with a third version (Euler) in force. If we had tried to use  $\mathit{\mathbf{sfmath}}$  in this version we would have gotten an error message stating that this *(math alphabet identifier)* isn’t defined for the Euler version.<sup>7</sup>

<sup>7</sup> Actually we cheated a bit more in this article: we had to reset the  $\mathit{\mathbf{mathcode}}$  of certain characters because they are in different places in the Euler version. A few more details can be found in Don Knuth’s article [3]. However, this is not a real problem because such changes can be done in commands similar to  $\boldsymbol{\mathit{\mathbf{boldmath}}}$  if such incompatible versions are to coexist in real applications.

---

<sup>6</sup> This is done with the command  
 $\mathit{\mathbf{mathversion{version name}}}$ .

*LATEX* knows about three (*math alphabet identifier*)s. `\cal` will select calligraphic letters like  $ABCD$ , `\mathrm` will select upright roman letters for use in functions like  $\max_i$ , and finally `\mit` selects the default math italic alphabet.

Table 3: Predefined (*math alphabet identifiers*) in *LATEX*

If we are interested in a slanted shape we have to face a problem: there is no slanted shape in the bold extended series of the Computer Modern sans serif family. So, if we make the identifier known only in the normal version then it would produce an error message when encountered in the bold (or any other) version. Of course we can get by using always the same typeface in all versions. To make this task a bit easier there is also a \* variant of the `\newmathalphabet` command which takes three more arguments: the default values for family, series and shape for all math versions in which the alphabet identifier is not explicitly defined via an `\addtoversion` command. So our second example can be set up simply by stating

```
\newmathalphabet*{\sfslmath}{cmss}{m}{s1}
```

This would have the additional advantage that this math alphabet identifier is also allowed in math versions which are defined in style files or document styles (like the Euler version mentioned earlier). Any explicit `\addtoversion` command overwrites the defaults given by `\newmathalphabet*`; so, it might be a good idea always to specify default values.

Here we show the same formula as above, but this time in the Euler version and with `\sfslmath` instead of `\sfmath`:

$$\sum A_i = \tan \alpha$$

#### 4 Processing older documents

To typeset documents which are written with the old *LATEX* (i.e. with a format using the old font selection scheme) only the source line containing the `\documentstyle` command has to be changed. More exactly the 'oldfont' option must be added to the list of document style options if the new font selection scheme is in force.<sup>8</sup>

#### 5 Setting up a new format

This section is written for people called 'local wizards' by the *LATEX* manual, which simply refers to the (poor) guys who are always being pestered if

<sup>8</sup> This means that it is the default (see next section).

things do not work.<sup>9</sup> If you are using *LATEX* on your own PC you might have to read this section, too, even if you don't feel like being a wizard.

#### 5.1 Preparations

Before generating a new format it is necessary to rename a few files. This enables you to customize the format to the special needs of your site.

`lfonts.tex` First of all you should rename the file `lfonts.tex` (supplied with the standard distribution of *LATEX*); otherwise you will always end up with an old format. Call it, say, `lfonts_ori`.

`hyphen.tex` Another file which should probably be renamed is `hyphen.tex` (the original American `\patterns` from Don Knuth) because this enables you to insert your favourite `\pattern` package when *IniTeX* is asking for this file. This might even be useful if you use *TeX* version 3.0 which is multilingual (assuming that your computer has only a limited memory).

##### 5.1.1 Preloading Fonts

Now you have to decide which fonts to preload in your format. Unlike the old font selection scheme of *LATEX*, where only preloaded fonts could be used in math applications (like subscripts etc.), the new font selection scheme poses no restriction at all; documents will always come out the same. So you have to take your pick by weighing the two conflicting principles:

- Preloading often used fonts might make your *TeX* run a bit faster.
- Using more load-on-demand fonts will make your format much more flexible, because you can switch to different families far more easily. After all, there is an upper limit to the number of fonts *TeX* can use in one run and every preloaded font will count even if it is never accessed.

<sup>9</sup> YOU might belong to this group!

On the PC at home we nowadays always use formats with only 5 fonts preloaded.<sup>10</sup> We don't think that TeX is actually running much more slowly than before.

Together with the new font selection scheme two files preload.min and preload.ori are distributed. The first one will preload next to nothing while the second will preload the same fonts as the old lfonts.tex. You can copy either of these files to preload.tex and then change it if you want to preload some other fonts. But please make sure that you don't change one of the original files of the distribution.

### 5.1.2 Making more fonts available

Besides deciding which fonts to preload, you also have to tell the TeX system which external fonts are available and how they are organized in families, series, shapes and sizes. In short you have to set up internal tables giving informations like "family cmr, series b, shape n, size 10 is associated with the external font cmb10 but there is no font with similar characteristics in size 9". This is done with the \new@fontshape command, either in a style file (see 'concrete.sty' as an example) or when dumping a format.

Again two files fontdef.ori and fontdef.max are distributed. You can copy one of them to fontdef.tex. The file fontdef.ori defines all fonts which are necessary to run standard LATEX documents while fontdef.max also defines certain fonts from the AMSFonts collection. To make other font families available you can either append appropriate \new@fontshape definitions to fontdef.tex (again, leave the originals untouched!) or add them in a style file.<sup>11</sup> For a detailed description of how to set up new families with the \new@fontshape command, see [5] about the basic macros or one of the example files.

## 5.2 Running IniTeX

When setting up a new format one has to start IniTeX with lplain.tex as the input file. After displaying some progress report on the terminal,

<sup>10</sup> This is the absolute minimum. These fonts are accessed by lplain.tex and latex.tex when the format is generated.

<sup>11</sup> The latter alternative might be better if you use these fonts very rarely (e.g., at sites with many users) to avoid filling TeX's memory with unnecessary definitions.

lplain.tex will try to \input the files hyphen.tex and lfonts.tex.

As we said above, it seems a good idea to rename these files because, when TeX complains that it cannot find them and asks you to type in another file name, you get the chance to substitute your favourite hyphenation patterns without changing lplain.tex or copying something to hyphen.tex. The transcript file will show the name of the file used which is very useful to debug weird errors (later).

When the point is reached where TeX wants to read in lfonts.tex, you now have to specify 'lfonts.new'. This file will \input some other files. After processing them (which will take some time), IniTeX stops once more since it cannot find the file xxxlfont.sty. This is intentional; in this way you may now specify the desired default by entering one of the following file names:

oldlfont.sty If you choose this file, your format will be identical to the standard LATEX version 2.09 except that a few additional commands (like \normalshape) are available. Of course, documents or style options which explicitly refer to things like \tentt will produce error messages since such internal commands are no longer defined.<sup>12</sup> Nevertheless it is easy to fix the problem in such a case: if we know that \tentt referred to cmtt10, i.e. Computer modern typewriter normal at 10pt, we can define it as

```
\newcommand{\tentt}{\family{cmtt}
  \series{m}\shape{n}\size{10}{12pt}
  \selectfont}
```

Since we assume the 'oldlfont' option as default, where \tt resets series and shape, the definition could be shortened to

```
\newcommand{\tentt}{\size{10}{12pt}\tt}
```

To get the new way of font selection as described in the previous sections (e.g. where \tt simply means to switch to another family) you only have to add the 'newlfont' style option to the \documentstyle command in your document.

<sup>12</sup> By the way, such documents were at no time portable since Leslie Lamport stated that it was always permissible to customize lfonts.tex according to the local needs. Therefore this is *not* an incompatible change.

`newlfont.sty` This is just the counterpart to `oldlfont.sty`: it will make the new mechanism the default and you have to add 'oldlfont' as a style option if you want to process older documents which depend on the old mechanism.

`basefnt.tex` This file is similar to `newlfont.sty` but does not define the L<sup>A</sup>T<sub>E</sub>X symbol fonts. These fonts contain only a few characters which are also included in the AMS symbol fonts. Therefore we provided the possibility of generating a format which doesn't unnecessarily occupy one of the (only) sixteen math groups within one math version. Using this file you can easily switch to the old scheme (adding 'oldlfont' as an option), to the new scheme with L<sup>A</sup>T<sub>E</sub>X symbol fonts (using 'newlfont') or to the new scheme with additional AMS fonts by using either the style option 'amsfonts' (fonts only) or the style option 'amstex' (defining the whole set of A<sub>M</sub>S-T<sub>E</sub>X macros in a L<sup>A</sup>T<sub>E</sub>X like syntax).

We suggest using the `basefnt.tex` file since the new font selection scheme will be incorporated into L<sup>A</sup>T<sub>E</sub>X version 2.10, but on installations with many users it might be better to switch smoothly to the new font selection scheme by first using 'oldlfont' as a default.

Anyway, after reading the file chosen, T<sub>E</sub>X will continue by processing `latex.tex` and finally displaying the message "Input any local modifications here". If you don't dare to do so, use `\dump` to finish the run. This will leave you with a new `.fmt` file (to be put into T<sub>E</sub>X's format area) and the corresponding transcript file. It isn't a very good idea to delete this one because you might need it later to find out what you did when you dumped the format!

## 6 Remarks on the development of this interface

We started designing the new font selection scheme around April 1989. A first implementation was available after one month's work and thereafter the prototype version ran successfully for some months at a few sites in Germany and the UK. Frank's visit to Stanford as well as our work on the 'amstex' style option brought new aspects to our view. The result was a more or less complete redefinition of the L<sup>A</sup>T<sub>E</sub>X interface for this font mechanism. It was a long way from the first sketch (which was about five pages in Frank's notebook) to the current implementation of

the interface which now consists of nearly 2000 lines of code and about 4000 lines of internal documentation. The A<sub>M</sub>S-T<sub>E</sub>X project itself, which triggered this reimplemention, has about the same dimensions. Surely in such a huge software package one will find typos and bugs. But we hope that most of the bugs in the code are found by now. It is planned that the new font selection scheme will replace the old one in L<sup>A</sup>T<sub>E</sub>X version 2.10. We therefore hope that this pre-release which runs in version 2.09 will help to find all remaining problems so that the switch to version 2.10 will be without discomfort to the user.

## 7 Acknowledgements

During this project we got help from many people. A big 'thank you' to all of them, especially to Michael Downes from the AMS for his cooperation and help, to Stefan Lindner for his help with the Atari T<sub>E</sub>X and to Sebastian Rahtz for playing a willing guinea-pig. Finally we also want to thank Ron Whitney who did a marvelous job on all our articles so far. This time we posed some extra problems because he had to first make a new format in order to read how to make a new format.

## 8 List of distributed files

`lfonts.new` The new version of `lfonts.tex`, to be copied to a file of this name after the old `lfonts.tex` has been renamed.

`fontdef.ori` The font definitions for the computer modern fonts in the distribution by Donald E. Knuth. To be copied to `fontdef.tex` if this selection is to be used.

`fontdef.max` Complete font definitions for the computer modern fonts and the A<sub>M</sub>SFonts collections. To be copied to `fontdef.tex` if this selection is to be used.

`preload.ori` Preloads the same fonts as the old `lfonts.tex` does. To be copied to `preload.tex` if this is desired.

`preload.min` Preloads only the absolute minimum of fonts. To be copied to `preload.tex` if this is desired.

`fam.tex` The basic macros for the new font selection scheme. Automatically read by `lfonts.new`.

`latint.tex` L<sup>A</sup>T<sub>E</sub>X interface for the new font selection scheme. Automatically read by `lfonts.new`.

`setszie.tex` Size definitions for the L<sup>A</sup>T<sub>E</sub>X interface. Automatically read by `lfonts.new`.

**newlfont.sty** Selects new version of font selection for L<sup>A</sup>T<sub>E</sub>X.

**oldlfont.sty** Selects old version of font selection for L<sup>A</sup>T<sub>E</sub>X.

**basefnt.tex** Like newlfont.sty, but does not define the L<sup>A</sup>T<sub>E</sub>X symbol fonts.

**margid.sty** Style file that defines all (*math alphabet identifiers*) to have one argument. This is the default that is built in into the new font selection scheme. Therefore this style file is only necessary if the installation decided to load 'nomargid.sty' at dump time.

**nomargid.sty** In contrast to margid.sty, defines all (*math alphabet identifiers*) to switch to the alphabet. This style option is necessary if you want to be compatible to the old L<sup>A</sup>T<sub>E</sub>X syntax *in math mode only*.

**tracefnt.sty** Style file that allows the tracing of font usage. Use \tracingfonts with values 1 to 3 and watch what happens.

**syntonly.sty** Defines the \syntaxonly declaration. This can be used in the preamble of a document to suppress all output.

**amsfonts.sty** Defines the commands to select symbols from the AMSFonts collection.

**amsbsy.sty** Defines the \boldsymbol command.

**amssymb.sty** Defines additional A<sub>M</sub>S-T<sub>E</sub>X symbols.

**amstex.sty** Defines special A<sub>M</sub>S-T<sub>E</sub>X structures (like alignments in math mode) with L<sup>A</sup>T<sub>E</sub>X syntax.

**amstext.sty** Defines the A<sub>M</sub>S-T<sub>E</sub>X \text command.

**euscript.sty** Contains the definitions to use the Euler script fonts.

## References

- [1] Hugh Williamson, *Methods of Book Design*, Yale University Press, New Haven, London, Third Edition, 1985.
- [2] International Business Machines Corporation, *Font Object Content Architecture Reference*, First Edition, December 1988.
- [3] Donald E. Knuth, "Typesetting Concrete Mathematics," TUGboat 10, no. 1, 1989, pp. 31-36.
- [4] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik, *Concrete Mathematics*. Addison-Wesley, 1989.

- [5] Frank Mittelbach and Rainer Schöpf, "A New Font Selection Scheme for T<sub>E</sub>X Macro Packages," TUGboat 10, no. 2, 1989, pp. 222-238.

<p>◊ Frank Mittelbach Electronic Data Systems (Deutschland) GmbH Eisenstraße 56 D-6090 Rüsselsheim Federal Republic of Germany Bitnet: pzf5hz@drueds2</p>	<p>◊ Rainer Schöpf Institut für Theoretische Physik der Universität Heidelberg Philosophenweg 16 D-6900 Heidelberg Federal Republic of Germany Bitnet: BK4@DHDURZ1</p>
---	--

## Abstracts

Deutsche Kurzfassungen der  
TUGboat-Artikel  
[German Abstracts of TUGboat Articles]

Luzia Dietsche

T<sub>E</sub>X in Produktionsumgebung — Die Auswertung eines Fragebogens (E. M. Barnhart, S. 154)

Ungefähr vor einem Jahr erschien im TUGboat ein Fragebogen, der sich an T<sub>E</sub>X-Benutzer im Nicht-universitären Bereich wandte und die Anwendung von T<sub>E</sub>X und damit zusammenhängende Aspekte durch diesen Personenkreis behandelte. In diesem Artikel werden nun (zusammen jeweils mit den Fragen) in genauer Aufschlüsselung die Ergebnisse dieser Aktion vorgestellt. An der Aktion teilgenommen haben 60 Personen aus der gesamten T<sub>E</sub>X Welt.

Kodierung von Zeichensätzen (N. Beebe, S. 171)

Der Präsident der TUG stellt die diversen Probleme dar, die mit T<sub>E</sub>X 3.0 auf die T<sub>E</sub>X-Welt zukommen. Nicht nur die Frage, ob ISO-Norm *ja* oder *nein*, und wenn *ja*, welche der Normen, auch die Frage der Portabilität, der Kompatibilität, sowie der Trennung sind zu beachten. Für Interessierte an ISO8859 und ISO010646 existieren zwei Listen am Listserv in JHUVIM, in die sich alle mit *email*-Anschluß eintragen können.

**Auf dem Weg zu einer Standardisierung der neuen CM Fonts (J. S. Bień, S. 175)**

Der Vorschlag von Y. Haralambous aus *TUGboat* 10, no. 3, wie man den bisher ungenutzten Teil der CM Fonts standardisieren könnte, wird hier aus polnischer Sicht besprochen, vertieft und erweitert. Die Idee basiert darauf, ein vergrößertes CM Font Layout zu gestalten. Ein Beispiel für eine mögliche Anwendung der neuen Fonts wird mitgeliefert.

**Übungen für *TeX*: *The Program* (D. E. Knuth, S. 165)**

Prof. Knuth stellt eine Reihe von Übungen, Problemen und deren Lösungen vor, die er für einen von ihm im Frühjahr 1987 gehaltenen Kurs benötigte. Er sammelte und stellte sie für diejenigen zusammen, die *TeX*: *The Program* lernen wollen ohne in einen Kurs zu gehen.

**Kreisförmige Beweisführung: Einen Kreis setzen (und was man dazu braucht) (A. Hoenig, S. 183)**

Hier wird in dezidierter Form dargestellt, wie man mit *TeX* und *METAFONT* Buchstaben gedreht bis hin zur Kreisform darstellen kann. Dazu sind Modifikationen in *.mf* Quellecodes nötig, daraus resultierend jede Menge neuer Fonts und diverse Makros. Alle Änderungen, die dazu gebraucht werden, werden zusammen mit Beispielen und der dahinterstehenden Mathematik mitgeliefert.

**Einbinden von Graphiken in *TeX* mittels PostScript (G. Berendt, S. 190)**

Wer PostScript Graphiken auf PC erstellen kann und diese in *TeX* einbinden will, bekommt einen Treiber PTIPS vorgestellt, der es möglich macht, mit einer erweiterten *LATEX* picture Umgebung gezeichnete Bilder in *LATEX* Text-Dateien zu integrieren.

**Einbinden von Graphiken in *TeX* (G. Berendt, S. 190)**

Das Problem vieler PC-Benutzer ist es, daß *TeX* auf diesem Rechertyp nur einen sehr begrenzten Speicherplatz hat. Hier wird eine Lösung vorgestellt, mit der Bilder trotz der üblichen Grenzen gezeichnet und eingebunden werden können.

**Einbinden von Macintosh Graphiken in *LATEX* Dokumente (L. Schwer, S. 194)**

In diesem Artikel geht es um die Einbindung von Graphiken in *LATEX* auf einem Mac. Er soll für alle Neulinge einen Überblick geben. Da es keine

universelle Möglichkeit einer solchen Graphikeinbindung gibt, ist der Artikel so allgemein wie möglich gehalten. Er stützt sich auf DVIPS und PSFIG. Das benutzte Ausgabegerät ist ein Apple Laserwriter+. Um den Artikel zu verstehen, sind Kenntnisse von Mac, PostScript und *LATEX* von Nöten.

**Kombination von Graphiken und *TeX* auf einem PC mit Laser Drucker, Teil II (L. S. Pickrell, S. 200)**

In diesem Artikel soll eine früher aufgestellte Behauptung erhärtet werden, daß *TeX* einen ausgezeichneten Mechanismus zur Verfügung stellt, mit dem man Graphiken einbinden kann. Die dargestellte Methode zeigt die vollen Fähigkeiten von *TeX*, Graphiken genauso gut zu positionieren wie normalen Text. Zusammen mit den technischen Aspekten von diesem Projekt werden auch einige seiner Grenzen gezeigt. Außerdem soll über die eventuell vorhandenen Vorteile der Konvertierung von Graphiken in PK/TFM Files diskutiert werden. Einer der vielen Vorteile dieser Form der Graphikeinbindung ist es, daß sie auch zusammen mit PostScript Treibern verwendet werden kann.

**Für plain F&A: Eine neue Spalte im *TUGboat* (A. Hoenig, S. 212)**

Das Redaktionsteam vom *TUGboat* plant eine Frage- und Antworthecke für plain *TeX* für seine nächsten Ausgaben. Hierfür wird um rege Beteiligung gebeten. Beantwortet werden nur ganz konkrete Fragen zu plain *TeX*. *LATEX* Probleme werden auch weiterhin von J. Damrau beantwortet. Fragen à la "Ich brauche ein bestimmtes Makro. Wie geht das?" werden verständlicherweise nicht behandelt.

**Output Routinen: Beispiele und Techniken. Teil II: OTR Techniken (D. Salomon, S. 212)**

Im zweiten Teil seiner Serie über Output Routinen stellt der Autor einige Techniken vor, für die daselbe gilt, wie im ersten Teil: sie dürfen nicht einfach kopiert und angewandt werden. Im Gegenteil, sie sollen genau durchgearbeitet und an spezielle Probleme angepaßt werden. Der Schwerpunkt liegt diesmal darauf, einzelne Zeilen durch Output Routinen zu definieren, manipulieren und auszugeben. Des weiteren zeigt D. Salomon, wie man die OTR aktivieren kann oder einen zweimal durchlaufbaren Job generiert, der im ersten Schritt Informationen rauschreibt und im zweiten Schritt die Informationen wieder einliest.

**Listen in *TeX*'s Innerem (A. Jeffrey, S. 237)**

Es gibt in *TeX* zwar einen \item-Befehl, will man jedoch kompliziertere Listen benutzen und verwalten, muß man sich selbst weiterhelfen. Der Autor hat dazu eine Listenumgebung entwickelt, mit der es möglich ist, im Input in beliebiger Reihenfolge auf die Items einer Liste zu verweisen und sie in geordneter Reihenfolge wieder ausgeben zu lassen. Er hat in dem Artikel die genaue Entwicklung der Makros gepaart mit einer Menge Informationen über *TeX*-Makros beschrieben, so daß man gut verfolgen kann, welcher Schritt wofür von Nöten ist.

**Verbatim Mode zum schachteln (P. Taylor, S. 245)**

*verbatim.sty* in *LATEX* ist bekannt und ausgesprochen nützlich. Hier wird jetzt ein plain Makro vorgestellt, mit dem in *TeX* von einer *verbatim* Umgebung in die nächste geschachtelt werden kann, wobei zwischen *verbatim* und normaler Umgebung gewechselt wird. Dieser Schachtelung ist theoretisch keine Grenze gesetzt (allerdings dürfte irgendwann der stack-space zu Ende sein). Das Listing des Makros und ein Beispiel sind dem Artikel beigelegt.

**Easy Table (K. Ha, S. 250)**

Easy Table ist ein neues Programm, mit dem Tabellen jeder Couleur einfach und klar zu setzen sind. Trotz ca. 200 Kommandos ist es laut Autor leicht zu lernen und in *TeX* und *LATEX* einsetzbar. Wer näheres Interesse daran hat, die komplette Beschreibung und das Programmpaket selbst zu bekommen, muß sich mit dem Autor in Verbindung setzen.

**Bridge setzen mit *TeX* (K. v. d. Laan, S. 265)**

Es stehen jetzt plain Makros und eine Umgebung, mit der man Kartenverteilungen und Meldefolgen bei Bridge zeichnen kann, zur Verfügung. Dieses Paket stellt eine Fortsetzung und Erweiterung der in *TUGboat* 10, no. 1 vorgestellten *LATEX*-Styles dar. Zusätzlich sind Makros für die anspruchsvolle Darstellung eines ganzen Spielverlaufs in Vorbereitung. Beispiele für die Benutzung der Makros werden mitgeliefert.

**Die *LATEX* Ecke — Dblitem Makro (J. Damrau, S. 276)**

J. Damrau stellt einen Stylefile vor, mit dem einer Aufzählung zwei Argumente übergeben werden können: der gewünschte Text und das gewünschte

Trennzeichen. An ein solchen Makros oder Stylefiles besteht für eine Veröffentlichung jederzeit Interesse.

**35mm Folien mit *SLiTeX* (K. Yap, S. 279)**

*SLiTeX* ist auf den ersten Blick eine sehr nützliche Anwendung für alle, die regelmäßig Vorträge halten müssen. Allerdings nur für diejenigen, die einen Farbdrucker mit Druckmaßen größer als 35mm besitzen. Deshalb beschloß K. Yap, ein Konvertierprogramm für DVI in 35mm zu schreiben, gestützt auf die Treiberfamilie von N. Beebe. Das Ergebnis wird in die neue Treiberfamilie übernommen werden.

**Einfach Folien mit *LATEX* machen (G. Denk, S. 280)**

Eine weitere Möglichkeit, Folien herzustellen, wird mit *eslide.sty* angeboten. Allerdings nicht nach dem Prinzip von *SLiTeX*, sondern mit einer Stylefile Option zu *article.sty*. Ein bereits vorhandener Text kann durch diesen Zusatz mit wenigen Kommandos in gut lesbare Folien umgewandelt werden. Erhältlich ist die Source beim Autor.

**Eine neue Implementation der *LATEX* Umgebungen *verbatim* und *verbatim\** (Rainer Schöpf, S. 284)**

Diese Option zum *LATEX* \documentstyle Kommando stellt eine neue Implementation der *verbatim* und *verbatim\** Umgebungen dar. Ferner stellt sie die *comment* Umgebung zur Verfügung, die alle Eingabe zwischen \begin{comment} und dem nächsten \end{comment} überspringt. Schließlich wird noch das \verb Kommando neu definiert, um den Benutzer vor den Folgen eines vergessenen Begrenzungszeichens zu schützen.

◊ Luzia Dietsche  
 Rechenzentrum der Universität  
 Heidelberg  
 Im Neuenheimer Feld 293  
 D-6900 Heidelberg 1  
 Bitnet: X68@DHDURZ1

## Calendar

### 1990

May 21-25 Intensive Beginning/Intermed.  $\text{\TeX}$ ,  
University of Houston/Clear Lake,  
Houston, Texas

**TUG90 Conference**  
**Texas A & M University**  
**College Station, Texas**

Jun 11-15 Intensive Beginning/Intermed.  $\text{\TeX}$   
Jun 11-15 Advanced  $\text{\TeX}/\text{Macro Writing}$   
Jun 11-15 Intensive  $\text{\LaTeX}$   
Jun 11-15  $\text{\LaTeX}$  Style Files  
Jun 12-15 METAFONT  
Jun 13-15 Output Routines  
Jun 13-15 PostScript  
Jun 14-16 SGML  
Jun 18-20 TUG's 11<sup>th</sup> Annual Meeting  
Jun 21-22 Macro Writing  
Jun 21-23 Graphics in  $\text{\TeX}$

Jul 15 Papers from TUG Annual Meeting:  
Deadline for receipt of camera copy  
for *TUGboat* Proceedings issue.

**Providence College,**  
**Providence, Rhode Island**

Jul 16-20 Intensive Beginning/Intermed.  $\text{\TeX}$   
Jul 17-20 Beginning  $\text{\TeX}$   
Jul 23-27 Advanced  $\text{\TeX}/\text{Macro Writing}$   
Jul 24-27 Intermediate  $\text{\TeX}$

**Vanderbilt University, Nashville, Tennessee**

Jul 23-27 Intensive Beginning/Intermed.  $\text{\TeX}$   
Jul 30- Advanced  $\text{\TeX}/\text{Macro Writing}$   
Aug 3

**McGill University, Montréal, Québec**

Aug 6-10 Intensive Beginning/Intermed.  $\text{\TeX}$   
Aug 13-17 Advanced  $\text{\TeX}/\text{Macro Writing}$

**Rutgers University, Busch Campus,**  
**Piscataway, New Jersey**

Aug 6-10  $\text{\LaTeX}$  Style Files  
Aug 13-17 Advanced  $\text{\TeX}/\text{Macro Writing}$

**University of Illinois, Chicago, Illinois**

Aug 13-17 Intensive Beginning/Intermed.  $\text{\TeX}$   
Aug 14-17 Beginning  $\text{\TeX}$   
Aug 20-24 Intensive  $\text{\LaTeX}$   
Aug 21-24 Intermediate  $\text{\TeX}$

**California State University,**  
**Northridge, California**

Aug 20-24 Intensive Beginning/Intermed.  $\text{\TeX}$   
Aug 21-24 Beginning  $\text{\TeX}$   
Aug 27-31 Advanced  $\text{\TeX}/\text{Macro Writing}$   
Aug 28-31 Intermediate  $\text{\TeX}$

**University of Maryland,**  
**College Park, Maryland**

Aug 20-24 Intensive Beginning/Intermed.  $\text{\TeX}$   
Aug 27-31 Intensive  $\text{\LaTeX}$   
Sep 4-7 METAFONT

Aug 31 NTG-SGML Holland meeting  
Groningen, The Netherlands.  
For information, contact  
Kees van der Laan (Bitnet:  
CGL@RC.RUG.NL)

**$\text{\TeX}90$  Conference**  
**University College**  
**Cork, Ireland**

Sep 3-7 Intensive Beginning/Intermed.  $\text{\TeX}$   
Sep 3-5 Intensive  $\text{\LaTeX}$   
Sep 3-7 Intensive METAFONT  
Sep 5-7 SGML/ $\text{\TeX}$   
Sep 7-8 Advanced  $\text{\TeX}$   
Sep 10-13 TUG's 1<sup>st</sup> Conference in Europe  
Sep 14-15 Macro Writing  
Sep 14-15  $\text{\LaTeX}$  Style Files  
Sep 14-15 Graphics in  $\text{\TeX}$

- Sep 11      **TUGboat Volume 11,**  
**3<sup>rd</sup> regular issue:**  
 Deadline for receipt of manuscripts  
 (tentative).
- Sep 18-20 EP'90  
 National Institute of Standards  
 and Technology, Gaithersburg,  
 Maryland. For information,  
 contact Richard Furuta  
 ([furuta@brillig.umd.edu](mailto:furuta@brillig.umd.edu)).
- Oct 3-5 Seybold Computer Publishing  
 Conference, San Jose Convention  
 Center, San Jose, California.  
 For information, contact Seybold  
 Publications, West Coast Office  
 (213-457-5850).
- Oct 10-12 9<sup>th</sup> annual meeting, "Deutsch-  
 sprachige T<sub>E</sub>X-Interessenten";  
 DANTE e.V.: 3<sup>rd</sup> meeting, GWD,  
 Göttingen. For information, contact  
 Dr. Peter Scherber (Bitnet:  
[PSCHERB@DGOGWDG1](mailto:PSCHERB@DGOGWDG1)) or DANTE e.V.  
 (Bitnet: [DANTE@DHDURZ1](mailto:DANTE@DHDURZ1))
- Dec 6-8 European Publishing Conference,  
 Netherlands Congress Centre,  
 The Hague, Holland.  
 For information, contact Seybold  
 Publications, U.K. Office  
 ({44} 323 410561).

**1991**

- Jan 15      **TUGboat Volume 12,**  
**1<sup>st</sup> regular issue:**  
 Deadline for receipt of manuscripts  
 (tentative).
- Feb 20-22 10<sup>th</sup> annual meeting, "Deutsch-  
 sprachige T<sub>E</sub>X-Interessenten";  
 DANTE e.V.: 4<sup>th</sup> meeting,  
 Technical University of Vienna.  
 For information, contact  
 Dr. Hubert Partl (Bitnet:  
[Z3000PA@AWITUW01](mailto:Z3000PA@AWITUW01)) or DANTE e.V.  
 (Bitnet: [DANTE@DHDURZ1](mailto:DANTE@DHDURZ1))
- Apr 9        **TUGboat Volume 12,**  
**2<sup>nd</sup> regular issue:**  
 Deadline for receipt of manuscripts  
 (tentative).

- Sep 10      **TUGboat Volume 12,**  
**3<sup>rd</sup> regular issue:**  
 Deadline for receipt of manuscripts  
 (tentative).

For additional information on the events listed above, contact the TUG office (401-751-7760) unless otherwise noted.

**Production Notes**

Barbara Beeton

**Input and input processing**

Electronic input for articles in this issue was received by mail and on floppy disk.

Authors who had written articles previously for *TUGboat* typically submitted files that were fully tagged and ready for processing with the *TUGboat* macros—*tugboat.sty* for plain-based files and *ltugboat.sty* for those using *L<sup>A</sup>T<sub>E</sub>X*. (The macros—see the Authors' Guide, *TUGboat* 10, no. 3, pages 378-385—have been installed at *labrea.stanford.edu* and the other archives, and should be retrieved by prospective authors before preparing articles; for authors who do not have network access, the TUG office can provide the macros on diskette.)

Almost two-thirds of the articles, and about 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 *tugboat.sty* or *ltugboat.sty* as convenient. Most 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*.

Several articles required extra-special handling. The article by Mittelbach and Schöpf (p. 297) was set using a preliminary version of the new *L<sup>A</sup>T<sub>E</sub>X* font access technique which it describes. And the articles by Ha (p. 250) and Salomon (p. 212) used an experimental enhancement of the plain *TUGboat* macros that permits changing the number of columns in mid-page.

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 7 runs of *T<sub>E</sub>X* and 9 of *L<sup>A</sup>T<sub>E</sub>X*, using the page number file for reference.

The following articles were not prepared using *L<sup>A</sup>T<sub>E</sub>X*.

- Barbara Beeton, *Editorial comments*, page 153.
- Elizabeth Barnhart, *T<sub>E</sub>X in the production environment—questionnaire responses*, page 154.
- Donald Knuth, *Exercises for T<sub>E</sub>X: The Program*, page 165.
- Alan Hoenig, *Circular reasoning: typesetting on a circle, and related issues*, page 183.
- Bart Childs, *Data General site report*, page 206.
- Barbara Beeton, *Resources available to T<sub>E</sub>X users*, page 207.
- Ted Nieland, *DECUS T<sub>E</sub>X collection—submissions wanted*, page 211.
- Alan Hoenig, *Just plain Q&A*, page 212.

- David Salomon, *Output routines: Examples and techniques. Part II: OTR techniques*, page 212.
- Khanh Ha, *Easy Table*, page 250.
- Philip Taylor, *A nestable verbatim mode*, page 245.
- Kees van der Laan, *Typesetting bridge via T<sub>E</sub>X*, page 265.

## Output

The bulk of this issue was prepared on an IBM PC-compatible 386 using PC *T<sub>E</sub>X* and output on an APS-μ5 at the American Mathematical Society using resident CM fonts and additional downloadable fonts for special purposes.

The article by Lee S. Pickrell (cited above) required output to be prepared on an HP LaserJet II.

Only one item (other than advertisements) was received as camera copy: the figures for the *Output routines* tutorial by David Salomon (p. 212), which were prepared on a 300 dpi Apple LaserWriter.

The output devices used to prepare the advertisements were not usually identified; anyone interested in determining how a particular ad was prepared should inquire of the advertiser.

## New Publications and Software Available through the T<sub>E</sub>X Users Group, June 1990

*(These product descriptions were taken, for the most part, from the publishers' announcements.)*

### T<sub>E</sub>X for the Impatient

by Paul W. Abrahams, with Karl Berry and Kathryn A. Hargreaves

If you're eager to find fast answers to common *T<sub>E</sub>X* questions, your wait will soon be over. *T<sub>E</sub>X for the Impatient*, a practical handbook for *T<sub>E</sub>X*, will be available this July. Clear, concise, and accessible, this book is organized for easy retrieval of information. It's thoroughly indexed and carefully designed so you can learn by example. Plus, it is packed with explicit instructions, useful tips and techniques, and a wealth of lightly humorous and very illuminating examples. Features include:

- complete descriptions of *T<sub>E</sub>X* commands, arranged for lookup either by function or alphabetically;

- clear definitions of essential *T<sub>E</sub>X* concepts, collected in a separate chapter so that the command descriptions remain brief and accessible;
- explanations of common error messages and advice on solving problems that frequently arise;
- collection of useful macros (also available in electronic form).

*Addison-Wesley Publishing Co., Reading, Mass.*, 1990, 384 pp.

### L<sup>A</sup>T<sub>E</sub>X for Engineers and Scientists

by David J. Buerger

Your comprehensible guide to *L<sup>A</sup>T<sub>E</sub>X*: Coping with even the most complex multiline equations—well beyond the scope of most computerized publishing systems—is a simple matter when you combine the high-powered functionality of *L<sup>A</sup>T<sub>E</sub>X* with this guide. With *L<sup>A</sup>T<sub>E</sub>X*, scientists, engineers, and other professionals can produce technical documents to the highest professional typeset standards. This

fast, easy-to-use primer quickly brings newcomers to L<sup>A</sup>T<sub>E</sub>X up to complete mastery of this powerful new software tool's most sophisticated features. A pleasure to use, this book clearly spells out how to:

- change fonts and type sizes at will for the most impressive, professional results;
- organize book-length documents with the highest levels of editorial integrity;
- create footnotes, cross-references, bibliographies, and indexes automatically;
- generate presentation-quality tables and figures with a publisher's precision;
- handle two-column documents in the style of professional proceedings and journals.

To further assist you, the author provides exercises (with answers), sample input files, a table of mathematical symbols, a convenient glossary of L<sup>A</sup>T<sub>E</sub>X forms, and special help with deciphering error messages.

*McGraw-Hill, New York, 1990, xvii + 199 pp.*

### Proceedings

#### Third European T<sub>E</sub>X Conference

#### T<sub>E</sub>X88—Exeter, August 1988

#### T<sub>E</sub>X: Applications, Uses, Methods

Malcolm Clark, editor

#### Table of Contents:

- Peter Abbott: UKT<sub>E</sub>X and the Aston archive.  
 Graham Asher: Type and set: T<sub>E</sub>X as the engine of a friendly publishing system.  
 Anne Brüggemann-Klein and Derick Wood:  
     Drawing trees nicely with T<sub>E</sub>X.  
 Lance Carnes and William S. Kaster: T<sub>E</sub>X device drivers today.  
 Francis J. Cave: The notation and structure of mathematical texts and their representation within electronic documentation systems.  
 Malcolm Clark and Cathy Booth: Whither T<sub>E</sub>X?  
     Why has T<sub>E</sub>X not taken over the world . . . ?  
 Frank R. Drake, John Derrick, and Laurent Siebenmann: Sweet-t<sub>E</sub>X, a report.  
 Roger Gawley: T<sub>E</sub>X in the mainframe world—the Durham experience.  
 Klaus Guntermann and Joachim Schrod: High quality DVI drivers.  
 Alois Heinz: Including pictures in T<sub>E</sub>X.  
 Alan Hoenig: An introduction to T<sub>E</sub>X for new users.  
 Alan Hoenig: Line-oriented layout with T<sub>E</sub>X.  
 Bogusław Jackowski, Tomasz Hołdys, and Marek Ryćko: With T<sub>E</sub>X to the Poles.  
 Susanne Lachmann: PROT<sub>E</sub>X: Integration of text, graphics and images.

Rod Mulvey: The Cambridge T<sub>E</sub>X-to-Type service.  
 Bill Noble and Rachel Ganz: T<sub>E</sub>X and good design—are they compatible?

A. C. Norris and A. L. Oakley: Electronic publishing and chemical text processing.

Peter J. Olivier: Publishing 'exotic' documents with ExoT<sub>E</sub>X.

Victor Ostromoukhov: METAFONT versus PostScript.

Hubert Partl: German T<sub>E</sub>X.

Gerlinde Petersen: LinoT<sub>E</sub>X: professional electronic publishing.

Sebastian P. Q. Rahtz: A survey of picture-drawing in L<sup>A</sup>T<sub>E</sub>X.

Michael Ramek: Chemical structure formulæ and x/y diagrams with T<sub>E</sub>X.

Rainer Rupprecht: Using menudriven T<sub>E</sub>X under MVS.

Richard O. Simpson: Nontraditional uses of METAFONT.

Thomas Stadler and Tibor Tscheke: An environment for T<sub>E</sub>X-output with original Monotype fonts.

Jan van Knippenberg: Quality printing of T<sub>E</sub>X DVI files.

Jörg Winckler: T<sub>E</sub>X-fonts in image generation software.

*Ellis Horwood, Chichester, 1990, 271 pp.*

### 1989 Conference Proceedings

#### T<sub>E</sub>X Users Group

#### Stanford University, August 1989

#### Ten Years of T<sub>E</sub>X and METAFONT

Christina Thiele, editor

#### Table of Contents:

Editor's Introduction

#### Keynote Address

Donald E. Knuth: The Errors of T<sub>E</sub>X

#### Font Forum

Doug Henderson: Introduction to METAFONT

Neenie Billawala: Opening Pandora's Box

Alan Hoenig: Fractal images with T<sub>E</sub>X

Don Hosek: Design of oriental characters with METAFONT

Bob Batzinger: Thai Languages and METAFONT

John D. Hobby: A METAFONT-like system with PostScript output

Ralph E. Youngen, Daniel C. Latterner, and William B. Woolf: Migration from Computer Modern fonts to Times fonts

Arvin C. Conrad: Fine typesetting with T<sub>E</sub>X using native Autologic fonts

*TeX Training*

- Michael Doob: Of the computer scientist, by the computer scientist, for the computer scientist  
 Hope Hamilton: Mastering *TeX* with templates  
 Anita C. Hoover: Using WordPerfect 5.0 to create *TeX* and *LATeX* documents  
 Robin L. Kubek: *TeX* for the word processing operator  
 Jo Ann Rattey-Hicks: *TeX* and its versatility in office production

*General Applications*

- Max Díaz: *TeX* in México  
 James Haskell, Wally Deschene and Alan Stolleis: *TeX* for 30,000  
 Alan Wittbecker: *TeX* enslaved

*Graphics Applications*

- Tom Renfrow: Methodologies for preparing and integrating PostScript graphics  
 Rolf Olejniczak-Burkert: *texpic*—design and implementation of a picture graphics language in *TeX à la pic*

*Database Applications*

- William B. Woolf and Daniel C. Latterner: *TeX* at *Mathematical Reviews*  
 Jørgen L. Pind: Lexicography with *TeX*

*General Information*

- Malcolm Clark: Olde Worlde *TeX*  
 Peter Abbott: The UK*TeX* Archive at University of Aston

*TeX Tools*

- Frank Mittelbach and Rainer Schöpf: With *LATeX* into the nineties  
 Andrew Marc Greene: *TeX*reation—Playing games with *TeX*'s mind  
 Bill Cheswick: A permuted index for *TeX* and *LATeX*  
 Steve Sydoriak: *LATeX* memos and letters  
 Gary Benson, Debi Erpenbeck and Janet Holmes: Inserts in a multiple-column format  
 Mary McClure: *TeX* macros for COBOL syntax diagrams

- Brad L. Halverson and Don L. Riley: Creating an efficient and workable PC interface for *TeX*

*TeX Users Group, Providence, R.I., 1989*  
 (published as *TUGboat* 10, no. 4).

**Vector*TeX***

retains all the advantages of *TeX* plus:

- saves megabytes of storage—entire V*TeX* fits on one floppy;
- instantly generate any font in any size and in any variation from 5 to 90 points;

- standard font effects include compression, slant, smallcaps, outline and shading. New: shadow;
- discover the universe of MicroPress professional typefaces: not available for any other *TeX*.

Includes the V*TeX* typesetter, 10 scalable typefaces, VVIEW (arbitrary magnification on EGA, CGA, VGA, Hercules, AT&T), VLASER (HP LaserJet), VPOST (PostScript), VDOT (Epson, Panasonic, NEC, Toshiba, Proprinter, Star, DeskJet) and manuals.

*MicroPress, Inc., Forest Hills, N.Y.*

**AP-*TeX* Fonts**

provide the quality of Adobe PostScript fonts for your *TeX* documents and non-PostScript printer. If you use any brand of *TeX* with an HP LaserJet or DeskJet printer, the AP-*TeX* fonts add a wealth of attractive typefaces identical to the popular PostScript extended font families. By de-crypting the Adobe coding it is possible to exactly translate the PostScript fonts into *TeX* font bit map and metric files. These translated fonts include the renowned Adobe "hints," which render the smaller point sizes of the fonts with remarkable clarity on laser and ink-jet printers. The fonts use the *TeX* character set encoding and font metrics, including full kerning and ligature programs. The AP-*TeX* fonts, supplied on ten 360K 5-1/4" PC floppy disks, contain 35 typefaces in pk format (including *TeX* font metric (tfm) files) for 300 dots/inch laser and ink-jet printers. The fonts included are identical to the Adobe PostScript implementations of the trade names and samples shown on page 147, *TUGboat* 11, no. 1 (1990). The point sizes for each typeface included are the *TeX* sizes 5, 6, 7, 8, 9, 10, 11, 12, 14.4, 17.3 20.7, and 24.9 points. Headline styles (equal to Times Roman, Helvetica, and Palatino, all in bold) also are included at 29.9, 35.8, 43.0, 51.6, 61.9, and 74.3 points.

*The Kinch Computer Co., Ithaca, New York*

**CAPTURE**

is the graphics solution for PC-based *TeX*. It places graphics in *TeX* documents produced on IBM PC systems (and compatibles) with Hewlett-Packard LaserJet printers. It doesn't require PostScript.

CAPTURE is designed for *TeX*. It carefully removes all 28 LaserJet control codes that disrupt *TeX*. It has been tested with PCT*TeX*,  $\mu$ *TeX*, and *TeXplus*. It "captures" the graphics generated by any application program, including "paint" programs, circuit design, CAD, scientific data plotters, optic design, terminal emulators, clip art, spreadsheets, databases—anything that supports

the LaserJet. It supports PostScript. Graphics can be converted to the pk/tfm format of TeX and used with PostScript drivers. View your graphics on screen previewers. Graphics can be manipulated by TeX. Do anything with graphics you can do with type; graphics and text are handled the same.

*Micro Programs, Inc., Syosset, N.Y.*

#### **texpic**

by Rolf Olejniczak-Burkert

*texpic* is a TeX implementation of a graphics language similar to Kernighan's *troff* preprocessor *pic*. Many features of the original *pic* are supported,

including a variety of graphical objects (boxes, circles, ellipses, lines, arrows and others), directions of motion, controlling sizes of objects with variable and appropriate defaults, relative and absolute positioning of single objects or whole pictures (labels and corners are allowed), and much more. There are two significant enhancements. Objects adapt to the size of their contents; that is, a circle may contain a table with mathematical equations, a box may contain the circle, etc. *texpic* objects and TeX or L<sup>A</sup>T<sub>E</sub>X commands may be combined at will.

*Micro Programs, Inc., Syosset, N.Y.*

#### **Russians Visit TUG Headquarters**



From left to right: Ray Goucher and Karen Butler (TUG), Barbara Beeton (AMS/TUG), Irina Makhovaya, Irina Gorbunova and Andrei Smirnov (U.S.S.R.), Michael Downes (AMS) and Charlotte Laurendeau (TUG).

While on a 4-week visit to the American Mathematical Society in April to learn *AMS-Tex*, Irina Makhovaya of Mir Publishers, Moscow, Irina Gorbunova of Nauka Publishers & Booksellers, Moscow, and Andrei Smirnov of Leningrad University, spent several hours visiting with staff members at the TUG office in Providence, R. I. Discussions evolved around products and services TUG had to offer, formation of a TeX user association in the U.S.S.R.

and ways in which TUG can help them disseminate TeX in the U.S.S.R.

They were taken to an Italian restaurant for lunch, which was a new experience for each of them. An invitation has been extended to them to attend TeX90, Cork, Ireland, in September, where a "TeX Summit" with representatives of TeX user associations in Eastern and Western Europe will be held.

## Institutional Members

The Aerospace Corporation,  
*El Segundo, California*

Air Force Institute of Technology,  
*Wright-Patterson AFB, Ohio*

American Mathematical Society,  
*Providence, Rhode Island*

ArborText, Inc.,  
*Ann Arbor, Michigan*

ASCII Corporation,  
*Tokyo, Japan*

Aston University,  
*Birmingham, England*

Belgrade University,  
*Faculty of Mathematics,  
Belgrade, Yugoslavia*

Brookhaven National Laboratory,  
*Upton, New York*

Brown University,  
*Providence, Rhode Island*

California Institute of Technology,  
*Pasadena, California*

Calvin College,  
*Grand Rapids, Michigan*

Carleton University,  
*Ottawa, Ontario, Canada*

Carnegie Mellon University,  
*Pittsburgh, Pennsylvania*

Centre Inter-Régional de  
Calcul Électronique, CNRS,  
*Orsay, France*

College of William & Mary,  
Department of Computer Science,  
*Williamsburg, Virginia*

DECUS, L&T Special Interest  
Group, *Marlboro, Massachusetts*

Department of National Defence,  
*Ottawa, Ontario, Canada*

Digital Equipment Corporation,  
*Nashua, New Hampshire*

Edinboro University  
of Pennsylvania,  
*Edinboro, Pennsylvania*

Emerson Electric Company,  
*St. Louis, Missouri*

Environmental Research  
Institute of Michigan,  
*Ann Arbor, Michigan*

European Southern Observatory,  
*Garching bei München,  
Federal Republic of Germany*

Fermi National Accelerator  
Laboratory, *Batavia, Illinois*

Fordham University,  
*Bronx, New York*

Försvarets Materielverk,  
*Stockholm, Sweden*

General Motors  
Research Laboratories,  
*Warren, Michigan*

Geophysical Company  
of Norway A/S,  
*Stavanger, Norway*

GKSS, Forschungszentrum  
Geesthacht GmbH,  
*Geesthacht, Federal Republic of  
Germany*

Grinnell College,  
Computer Services,  
*Grinnell, Iowa*

Harvard University,  
Computer Services,  
*Cambridge, Massachusetts*

Hatfield Polytechnic,  
Computer Centre,  
*Herts, England*

Hewlett-Packard Co.,  
*Boise, Idaho*

Hughes Aircraft Company,  
Space Communications Division,  
*Los Angeles, California*

IBM Corporation,  
Scientific Center,  
*Palo Alto, California*

Institute for Advanced Study,  
*Princeton, New Jersey*

Institute for Defense Analyses,  
Communications Research  
Division, *Princeton, New Jersey*

Intevep S. A., *Caracas, Venezuela*

Iowa State University,  
*Ames, Iowa*

Kuwait Institute for  
Scientific Research,  
*Safat, Kuwait*

The Library of Congress,  
*Washington D.C.*

Los Alamos National Laboratory,  
University of California,  
*Los Alamos, New Mexico*

Louisiana State University,  
*Baton Rouge, Louisiana*

Marquette University,  
Department of Mathematics,  
Statistics and Computer Science,  
*Milwaukee, Wisconsin*

Massachusetts Institute  
of Technology,  
Artificial Intelligence Laboratory,  
*Cambridge, Massachusetts*

Mathematical Reviews,  
American Mathematical Society,  
*Ann Arbor, Michigan*

Max Planck Institut  
für Mathematik,  
*Bonn, Federal Republic of Germany*

Max Planck Institute Stuttgart,  
*Stuttgart, Federal Republic of  
Germany*

McGill University,  
*Montréal, Québec, Canada*

Michigan State University,  
Mathematics Department,  
*East Lansing, Michigan*

National Cancer Institute,  
*Frederick, Maryland*

National Research Council  
Canada, Computation Centre,  
*Ottawa, Ontario, Canada*

Naval Postgraduate School,  
*Monterey, California*

New Jersey Institute of  
Technology, *Newark, New Jersey*

New York University,  
Academic Computing Facility,  
*New York, New York*

Nippon Telegraph &  
Telephone Corporation,  
Software Laboratories,  
*Tokyo, Japan*

- Northeastern University,  
Academic Computing Services,  
*Boston, Massachusetts*
- Norwegian Pulp & Paper  
Research Institute,  
*Oslo, Norway*
- Pennsylvania State University,  
Computation Center,  
*University Park, Pennsylvania*
- Personal T<sub>E</sub>X, Incorporated,  
*Mill Valley, California*
- Princeton University,  
*Princeton, New Jersey*
- Promis Systems Corporation,  
*Toronto, Ontario, Canada*
- Peter Isaacson Publications,  
*Victoria, Australia*
- Purdue University,  
*West Lafayette, Indiana*
- Queens College,  
*Flushing, New York*
- RE/SPEC, Inc.,  
*Rapid City, South Dakota*
- Rice University,  
Department of Computer Science,  
*Houston, Texas*
- Rogaland University,  
*Stavanger, Norway*
- Ruhr Universität Bochum,  
Rechenzentrum,  
*Bochum, Federal Republic of Germany*
- Rutgers University, Hill Center,  
*Piscataway, New Jersey*
- St. Albans School,  
*Mount St. Alban, Washington,  
D.C.*
- Sandia National Laboratories,  
*Albuquerque, New Mexico*
- SAS Institute,  
*Cary, North Carolina*
- I.P. Sharp Associates,  
*Palo Alto, California*
- Smithsonian Astrophysical  
Observatory, Computation Facility,  
*Cambridge, Massachusetts*
- Software Research Associates,  
*Tokyo, Japan*
- Sony Corporation,  
*Atsugi, Japan*
- Space Telescope Science Institute,  
*Baltimore, Maryland*
- Springer-Verlag,  
*Heidelberg, Federal Republic of Germany*
- Stanford Linear  
Accelerator Center (SLAC),  
*Stanford, California*
- Stanford University,  
Computer Science Department,  
*Stanford, California*
- Stefan Ram, Programming and  
Trade, *Berlin, Federal Republic of Germany*
- Syracuse University,  
*Syracuse, New York*
- Talaris Systems, Inc.,  
*San Diego, California*
- TECOGRAF Software,  
*Milan, Italy*
- Texas A & M University,  
Department of Computer Science,  
*College Station, Texas*
- Texcel, *Oslo, Norway*
- TRW, Inc., *Redondo Beach,  
California*
- Tufts University,  
*Medford, Massachusetts*
- TV Guide, *Radnor, Pennsylvania*
- TYX Corporation,  
*Reston, Virginia*
- UNI-C, *Aarhus, Denmark*
- Universidad Sevilla,  
*Sevilla, Spain*
- Universidade de Coimbra,  
*Coimbra, Portugal*
- Università degli Studi Milano,  
Istituto di Cibernetica,  
*Milan, Italy*
- University College,  
*Cork, Ireland*
- University of Alabama,  
*Tuscaloosa, Alabama*
- University of British Columbia,  
Computing Centre,  
*Vancouver, British Columbia,  
Canada*
- University of British Columbia,  
Mathematics Department,  
*Vancouver, British Columbia,  
Canada*
- University of Calgary,  
*Calgary, Alberta, Canada*
- University of California,  
Division of Library Automation,  
*Oakland, California*
- University of California, Berkeley,  
Computer Science Division,  
*Berkeley, California*
- University of California, Berkeley,  
Space Astrophysics Group,  
*Berkeley, California*
- University of California, Irvine,  
Department of Mathematics,  
*Irvine, California*
- University of California, Irvine,  
Information & Computer Science,  
*Irvine, California*
- University of California, Los Angeles, Computer  
Science Department Archives,  
*Los Angeles, California*
- University of California, San Diego, *La Jolla, California*
- University of Canterbury,  
*Christchurch, New Zealand*
- University of Chicago,  
Computing Organizations,  
*Chicago, Illinois*
- University of Chicago,  
*Chicago, Illinois*
- University of Crete,  
Institute of Computer Science,  
*Heraklio, Crete, Greece*
- University of Delaware,  
*Newark, Delaware*
- University of Exeter,  
Computer Unit,  
*Exeter, Devon, England*
- University of Glasgow,  
Department of Computing Science,  
*Glasgow, Scotland*

University of Groningen,  
*Groningen, The Netherlands*

University of Illinois at Chicago,  
Computer Center,  
*Chicago, Illinois*

University of Kansas,  
Academic Computing Services,  
*Lawrence, Kansas*

University of Maryland,  
Department of Computer Science,  
*College Park, Maryland*

University of Maryland  
at College Park,  
Computer Science Center,  
*College Park, Maryland*

University of Massachusetts,  
*Amherst, Massachusetts*

Université de Montréal,  
*Montréal, Québec, Canada*

University of Oslo,  
Institute of Informatics,  
*Blindern, Oslo, Norway*

University of Oslo,  
Institute of Mathematics,  
*Blindern, Oslo, Norway*

University of Ottawa,  
*Ottawa, Ontario, Canada*

University of Salford,  
*Salford, England*

University of Southern California,  
Information Sciences Institute,  
*Marina del Rey, California*

University of Stockholm,  
Department of Mathematics,  
*Stockholm, Sweden*

University of Texas at Austin,  
*Austin, Texas*

University of Vermont,  
*Burlington, Vermont*

University of Washington,  
Department of Computer Science,  
*Seattle, Washington*

University of Western Australia,  
Regional Computing Centre,  
*Nedlands, Australia*

University of Wisconsin,  
Academic Computing Center,  
*Madison, Wisconsin*

Uppsala University,  
*Uppsala, Sweden*

USDA Forest Service,  
*Washington, D.C.*

Vereinigte Aluminium-Werke AG,  
*Bonn, Federal Republic of Germany*

Villanova University,  
*Villanova, Pennsylvania*

Vrije Universiteit,  
*Amsterdam, The Netherlands*

Washington State University,  
*Pullman, Washington*

Widener University,  
Computing Services,  
*Chester, Pennsylvania*

John Wiley & Sons, Incorporated,  
*New York, New York*

Worcester Polytechnic Institute,  
*Worcester, Massachusetts*

Yale University, Computer Center,  
*New Haven, Connecticut*

Yale University,  
Department of Computer Science,  
*New Haven, Connecticut*



## Publishing Services



### From the Basic

The American Mathematical Society can offer you a basic **TEX** publishing service. You provide the DVI file and we will produce typeset pages using an Autologic APS Micro-5 phototypesetter. The low cost is basic too: only \$5 per page for the first 100 pages; \$2.50 per page for additional pages, with a \$30 minimum. Quick turnaround is important to you and us ... a manuscript up to 500 pages can be back in your hands in just one week or less.

### To the Complex

As a full service **TEX** publisher, you can look to the American Mathematical Society as a single source for all your publishing needs.

**Macro-Writing**

**TEX Problem Solving**

**Autologic Fonts**

**Keyboarding**

**Art and Pasteup**

**Camera Work**

**Printing**

**Binding**

For more information or to schedule a job, please contact Regina Girouard, American Mathematical Society, P.O. Box 6248, Providence, RI 02940 or call 401-455-4060 or 800-321-4AMS in the continental U.S.

### 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 [ ] Address: _____
Bus. [ ] _____
_____
_____

Qty	1990 Membership/TUGboat Subscription (Jan.-Dec.)											Amount
	New (first-time): [ ] \$35.00 each; students [ ] \$25.00 each											
	Renewal: [ ] \$45.00; [ ] \$35.00 - reduced rate if renewed before February 1, 1990											
	Mailing charges per subscription: Canada/Mexico - \$5; Europe - \$10; Other Countries - \$15											
	TUGboat back volumes	1980	1981	1982	1983	1984	1985	1986	1987	1988	1989	
	Circle volume(s) desired:	v. 1	v. 2	v. 3	v. 4	v. 5	v. 6	v. 7	v. 8	v. 9	v. 10	
		\$18	\$50	\$35	\$35	\$35	\$50	\$50	\$50	\$50	\$75	

Issues of TUGboat will be shipped via air service 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:

My installation can offer the following software or technical support to TUG:

[ ] Pascal

[ ] C

[ ] other (describe) \_\_\_\_\_

From whom obtained: \_\_\_\_\_

Please list high-level TeX users at your site who would not mind being contacted for information; give name, address, and telephone.

Hardware on which TeX is used:

Computer(s)	Operating system(s)	Output device(s)
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____

# VECTOR TEX

B  
T  
w or  
g  
C & H  
E A  
M

## TEX FOR THE 90'S

**Are you still  
struggling with  
PXL's, PK's or GF's?  
Move on to scalable  
fonts:**

- Save megabytes of storage—entire VT<sub>E</sub>X fits on one floppy.
- Instantly generate any font in any size and in any variation from 5 to 100 points.
- Standard font effects include compression, slant, smallcaps, outline, shading and shadow.  
New: landscape.
- Discover the universe of MicroPress Font Library professional typefaces: not available from any other T<sub>E</sub>X vender.

List price \$399      **Introductory offer \$299**

Includes the VT<sub>E</sub>X typesetter (superset of T<sub>E</sub>X), 10 scalable typefaces, VVIEW (arbitrary magnification on EGA, CGA, VGA, Hercules, AT&T), VLASER (HP LaserJet), VPOST (PostScript), VDOT (Epson, Panasonic, NEC, Toshiba, Proprinter, Star, DeskJet) and manuals.

Introductory offer expires on September 1, 1990. S/H add \$5. COD add \$5. WordPerfect Interface add \$100. Site licenses available. Dealers' inquiries welcome. Professional typefaces available for older implementations of T<sub>E</sub>X.



**MicroPress Inc.**

67-30 Clyde Street, #2N, Forest Hills, NY 11375  
Tel: (718) 575-1816 Fax: (718) 575-8038

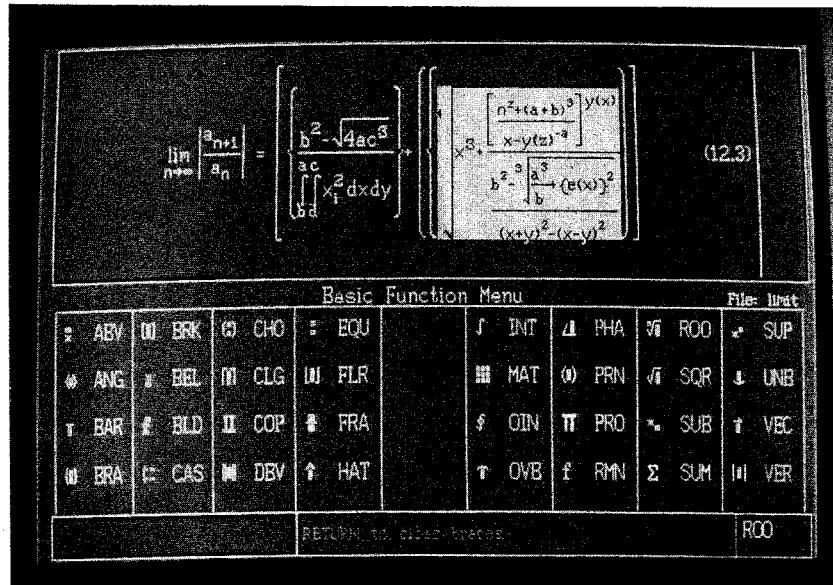
## TEX EDITION of MathEdit

This powerful new equation editor can  
be used to create equations for  $\text{\TeX}$ .  
For IBM PC

# MathEdit

- \* Easy to Use Version 2.0
- \* Menu driven so no codes need to be learned
- \* High-quality  $\text{\TeX}$  printing

**WYSIWYG →**  
View your equation as  
you create it. Then  
insert into your  $\text{\TeX}$   
document with one  
command.



TEX Edition ONLY \$129.00

Professional Edition \$199.00

Shipping: \$4 (U.S.A.), \$25 (Canada), \$35 (Overseas)

VISA, Mastercard and University and Government P.O.'s accepted.

**K-TALK®**  
COMMUNICATIONS

30 West First Avenue  
Columbus, Ohio 43201

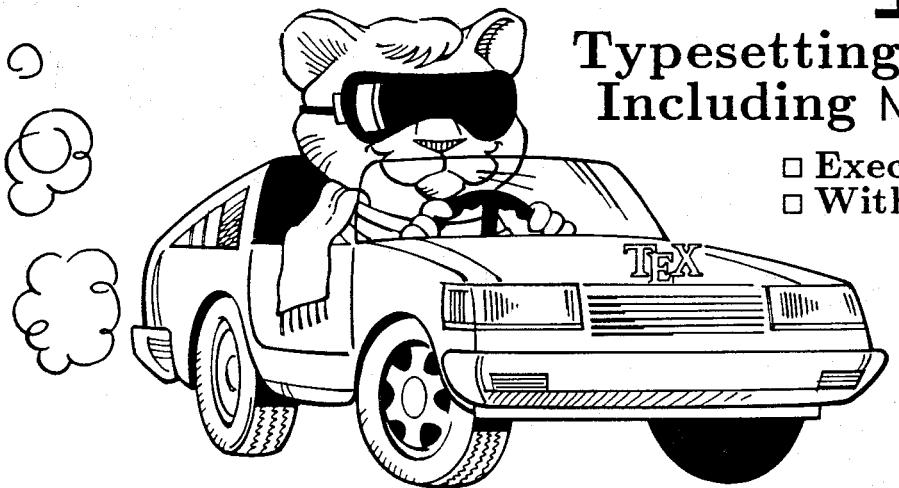
(614) 294-3535

FAX (614) 294-3704

# TurboTEX

**Typesetting Software  
Including METAFONT**

- Executables \$150
- With source \$300



TurboTeX Release 3.0 software brings you the latest TeX 3.0 and METAFONT 2.0 standards: preloaded plain TeX, L<sup>A</sup>T<sub>E</sub>X, *AM<sup>S</sup>-TeX* and *AM<sup>S</sup>-L<sup>A</sup>T<sub>E</sub>X*, and plain METAFONT interfaced to CGA/EGA/VGA/Hercules graphics; TRIP and TRAP certification; Computer Modern and L<sup>A</sup>T<sub>E</sub>X fonts, and printer drivers for HP LaserJet Plus/II/IIP, HP DeskJet, PostScript, and Epson LQ and FX dot-matrix printers. This wealth of software runs on your IBM PC (MS-DOS or OS/2), UNIX, or VAX/VMS system.

**■ Best-selling Value:** TurboTeX sets the standard for power and value among TeX implementations: one price buys a complete, commercially-hardened typesetting system. *Computer* magazine recommended it as "the version of TeX to have," *IEEE Software* called it "industrial strength," and thousands of satisfied users worldwide agree.

TurboTeX gets you started quickly, installing itself automatically under MS-DOS, and compiling itself automatically under UNIX. The 90-page User's Guide includes generous examples and a full index, and leads you step-by-step through installing and using TeX and METAFONT.

**■ Power Features:** TurboTeX breaks the 640K memory barrier under MS-DOS on any IBM-compatible PC with our virtual memory sub-system. Even without expanded memory hardware, 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 (like PICTTeX and *AM<sup>S</sup>-L<sup>A</sup>T<sub>E</sub>X* 2.0) 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.

**■ 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.

**■ Upgrades:** If you have TurboTeX Release 2.0, you can upgrade the executables for only \$40. If you have the source distribution, upgrade

both executables and source for \$80. Or, get either applicable upgrade free when you buy the AP-Tex fonts (see facing page) for \$200!

**■ No-risk trial offer:** Examine the documentation and run the PC TurboTeX for 10 days. If you are not satisfied, return it for a 100% refund or credit. (Offer applies to PC executables only.)

**■ Free Buyer's Guide:** Ask for the free, 70-page Buyer's Guide for more details on TurboTeX and dozens of TeX-related products: previewers, TeX-to-FAX and TeX-to-Ventura/Pagemaker translators, optional fonts, graphics editors, public domain TeX accessory software, books and reports.

## Ordering TurboTeX

Ordering TurboTeX is easy and delivery is fast, by phone, FAX, or mail. Terms: Check with order (free media and ground shipping in US), VISA, Mastercard (free media, shipping extra); Net 30 to well-rated firms and public agencies (shipping and media extra). Discounts available for quantities or resale. International orders gladly expedited via Air or Express Mail.

The Kinch Computer Company

PUBLISHERS OF TURBOTEX

501 South Meadow Street  
Ithaca, New York 14850 USA  
Telephone (607) 273-0222  
FAX (607) 273-0484

## AP-T<sub>E</sub>X Fonts

485 **TEX** fonts identical to  
Adobe PostScript Fonts for \$200

Get ready for the quality of Adobe PostScript fonts for your TeX documents and non-PostScript printer! If you use any brand of TeX with an HP LaserJet or DeskJet printer, the APTeX fonts from Kinch add a wealth of attractive typefaces identical to the popular PostScript extended font families.

By de-crypting the Adobe coding, we are able to exactly translate the PostScript fonts into TeX font bit map and metric files. These translated fonts include the renowned Adobe "hints," which render the smaller point sizes of the fonts with remarkable clarity on laser and ink-jet printers. The fonts use the TeX character set encoding and font metrics, including full kerning and ligature programs.

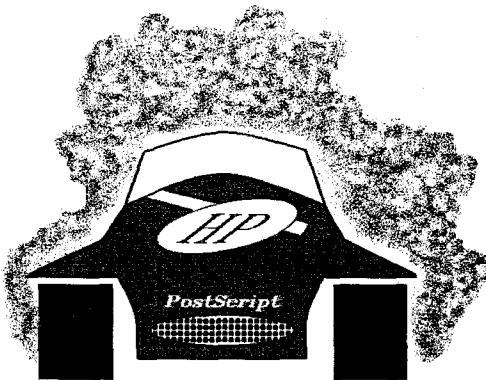
The AP-TEX fonts, supplied on ten 360K 5 1/4" PC floppy disks, contain 35 typefaces in PK format (including TEX font metric (TFM) files) for 300 dots/inch laser and ink-jet printers. The fonts included are identical to the Adobe Post-Script implementations of the trade names and samples shown at right. The point sizes for each typeface included are the TEX sizes 5, 6, 7, 8, 9, 10, 11, 12, 14.4, 17.3, 20.7, and 24.9 points. Headline styles (equal to Times Roman, Helvetica, and Palatino, all in bold) also are included at 29.9, 35.8, 43.0, 51.6, 61.9, and 74.3 points.

**The Kinch Computer Company**  
PUBLISHERS OF TURBOTEX  
**501 South Meadow Street**  
Ithaca, New York 14850  
Telephone (607) 273-0222  
FAX (607) 273-0484

Helvetica, Palatino, Times, and New Century Schoolbook are trademarks of Allied Linotype Co. ITC Avant Garde, ITC Bookman, ITC Zapf Chancery, and ITC Zapf Dingbats are registered trademarks of International Typeface Corporation. LaserJet and DeskJet are trademarks of Hewlett-Packard Corporation. PostScript is a registered trademark of Adobe Systems Incorporated. TeX is a trademark of the American Math Society. TurboTeX and AP-Tex are trademarks of Kinch Computer Company. Prices and specifications subject to change without notice. Revised February 8, 1990.

# Kinch

# TEX Plus 2.0 gives you more than just economy!



TEX Plus gives you the performance and features you need at a price you can afford.

Not only is our new 2.0 model faster (up to 40% faster than 1.8), we've also added support for EMS and an update to version 3.0 of the TEX language.

TEX Plus also includes our TEXWRITE editor, our TEXPRINT drivers for HP LaserJet and PostScript printers, and the standard macro packages and Computer Modern fonts. Plus we've added a driver for Epson/FX printers, and the SliTEX fonts—for only \$195.

## Free Upgrade

And that's not all! We'll provide you with a free upgrade to our 3.0 model when it's available. TEX Plus 3.0 will add a built-in previewer with support for CGA, EGA, VGA and Hercules adapters, and a Big TEX which provides 4 times the currently available TEX main memory. TEX Plus 3.0 will be priced at \$249, so take advantage of this offer and order now. (TEX Plus 3.0 will be available in mid-July).

## 386 Version

We'll be introducing a new version of TEX Plus—called TEX Pro—specially designed to take advantage of your 386 or 486 computer.

TEX Pro will include everything you would normally get with TEX Plus, including Big TEX, TEXWRITE, drivers and previewer, all designed to take advantage of the 386. TEX Pro will be available in mid-July and will be priced at \$295 with upgrades available from TEX Plus.

## Need a good driver?

If you just need a driver for a driver for your HP LaserJet Plus/Series II/IIP/III or PostScript printer, we've got just what you're looking for. Our TEXPRINT drivers are fast, full-featured, and reasonably priced at only \$129.

Both drivers offer support for landscape printing, inclusion of graphics (using packages such as CAPTURE), collating, odd or even page selection, and include a set of the Computer Modern and SliTEX fonts. With the HP driver you also get an HP soft font conversion utility, and with the PostScript driver you get an AFM to TFM conversion utility so that you can use both native and downloaded PostScript fonts.

Ask us about site licenses and our new LAN license. You'll be pleasantly surprised at how inexpensive it can be to put a complete TEX environment on your network.

To take one of our new models for a test drive just give us a call. All our products carry an unconditional 30-day money-back guarantee so you'll never be stuck with a lemon.



## Oregon House Software, Inc.,

Box 70, 12894 Rices Crossing Road,  
Oregon House, CA 95962  
(916) 692-1377

## Oregon House Software, Inc.,

Box 27057, 1395 Marine Drive,  
West Vancouver, B.C.,  
V7T 2X8 Canada  
(604) 926-0500

*Distributor inquiries welcome.*

TEX Plus is a trademark of the American Mathematical Society. TEX Plus is a trademark of Oregon House Software, Inc. All other product names are the trademarks or registered trademarks of their respective holders.

**Publishing Companion** translates

# WordPerfect

to

# T<sub>E</sub>X

It doesn't take a T<sub>E</sub>Xpert to use T<sub>E</sub>X.

With **Publishing Companion**, you can publish documents using T<sub>E</sub>X with **little or no T<sub>E</sub>X knowledge**. Your WordPerfect files are translated into T<sub>E</sub>X files, so anyone using this simple word processor can immediately begin typesetting their own documents!

And now, **Publishing Companion** translates **WordPerfect 5.0 and 5.1** files into T<sub>E</sub>X.

Retail Price .....	\$249.00
--------------------	----------

Academic Discount Price .....	\$199.00
-------------------------------	----------

For the power of T<sub>E</sub>X with the ease of a word processor, **Publishing Companion** is your "best friend" for desktop publishing.

For more information to place an order, call or write:



30 West First Ave., Suite 100  
Columbus, Ohio 43201  
(614) 294-3535  
FAX (614) 294-3704

**DESKTOP PUBLISHING HAS NEVER BEEN SIMPLER  
AND WILL NEVER BE THE SAME**

## Public Domain **TEX**

The public domain versions of **TEX** 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 **TEX** and **METAFONT**, the macro libraries for **AMS-TEX**, **LATETEX**, **SLiTEX** and **HP TEX**, 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. A portion of your tape cost is used to support development at Stanford University.

Compiled versions of **TEX** are available for DEC VAX/VMS, IBM CMS, IBM MVS and DEC TOPS systems. Systems using a standard format must compile **TEX** with a Pascal compiler.

### **TEX Order Form**

**TEX 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

Tape prices: \$92.00 for first tape, \$72.00 for each additional tape. Postage: allow 2 lbs. for each tape.

**Documents:**

	Price \$	Weight	Quantity
<b>T<small>EX</small>book (vol. A) softcover .....</b>	30.00	2	____
<b>T<small>EX</small>: The Program (vol. B) hardcover .....</b>	44.00	4	____
<b>METAFONT book (vol. C) softcover .....</b>	22.00	2	____
<b>METAFONT: The Program (vol. D) hardcover ...</b>	44.00	4	____
<b>Computer Modern Typefaces (vol. E) hardcover</b>	44.00	4	____
<b>L<small>AT</small>E<small>TEX</small> document preparation system .....</b>	30.00	2	____
<b>WEB language *</b> .....	12.00	1	____
<b>T<small>EX</small>ware *</b> .....	10.00	1	____
<b>BibT<small>EX</small> *</b> .....	10.00	1	____
<b>Torture Test for T<small>EX</small> *</b> .....	8.00	1	____
<b>Torture Test for METAFONT *</b> .....	8.00	1	____
<b>METAFONTware *</b> .....	15.00	1	____
<b>Metamarks *</b> .....	15.00	1	____

\* published by Stanford University

Orders from within California must add sales tax for your location.

Shipping charges: domestic book rate-no charge, domestic priority mail-\$1.50/lb, air mail to Canada and Mexico-\$2.00/lb, export surface mail (all countries)-\$1.50/lb, air mail to Europe, South America-\$5.00/lb, air mail to Far East, Africa, Israel-\$7.00/lb.

Purchase orders accepted. Payment by check must be drawn on a U.S. bank.

**Send your order to: Maria Code, DP Services, 1371 Sydney Drive, Sunnyvale, CA 94087  
FAX: 415-948-9388 Tel.: 415-735-8006.**

## The Wait Will Soon Be Over!

# TEX for the Impatient

Paul W. Abrahams

with Karl Berry and Kathryn A. Hargreaves

If you're eager to find fast answers to common **TEX** questions, your wait will soon be over. **TEX for the Impatient**, a practical handbook for **TEX**, will be available this July!

Clear, concise, and accessible, this book is organized for easy retrieval of information. It's thoroughly indexed and carefully designed so you can learn by example. Plus, **TEX for the Impatient** is packed with explicit instructions, useful tips and techniques, and a wealth of lightly humorous and very illuminating examples.

### Features include —

- Complete descriptions of **TEX** commands, arranged for lookup either by function or alphabetically
- Clear definitions of essential **TEX** concepts, collected in a separate chapter so that the command descriptions remain brief and accessible
- Explanations of common error messages and advice on solving problems that frequently arise
- Collection of useful macros (also available in electronic form)

Hurry! Follow the lead of the White Rabbit and add **TEX for the Impatient** to your set of **TEX** resources.

1990 (51375) 384 pp.

*Available wherever computer books are sold.*



Addison-Wesley Publishing Company

1 Jacob Way • Reading, MA 01867 • 617-944-3700

## Updated $\text{\TeX}$ Products from the American Mathematical Society



### $\mathcal{A}\mathcal{M}\mathcal{S}$ - $\text{\TeX}$ Version 2.0

$\mathcal{A}\mathcal{M}\mathcal{S}$ - $\text{\TeX}$ , the  $\text{\TeX}$  macro package that simplifies the typesetting of complex mathematics, has been updated to version 2.0.  $\mathcal{A}\mathcal{M}\mathcal{S}$ - $\text{\TeX}$  is intended to be used in conjunction with AMSFonts 2.0 (see below). However,  $\mathcal{A}\mathcal{M}\mathcal{S}$ - $\text{\TeX}$  can also be used without AMSFonts.  $\mathcal{A}\mathcal{M}\mathcal{S}$ - $\text{\TeX}$  is available on IBM or Macintosh diskettes—either format may be uploaded to many mainframe computers. **Prices:** \$30 list, \$27 AMS member. A free upgrade is available (until September 1, 1990), for those who have purchased a previous version.

### AMSFonts Version 2.0

AMSFonts 2.0 are designed for use with either  $\mathcal{A}\mathcal{M}\mathcal{S}$ - $\text{\TeX}$  2.0 or Plain  $\text{\TeX}$ . AMSFonts 2.0 **cannot** be used with previous versions of  $\mathcal{A}\mathcal{M}\mathcal{S}$ - $\text{\TeX}$ . Two distributions of fonts are available: one for use on PCs and mainframes (with any implementation of  $\text{\TeX}$ ), the other for use on a Macintosh with *Textures*. The fonts included on these distributions are:

Font Name	Point Sizes	Font Name	Point Sizes	
CMEX	CM Math Extension	7-9*	EUSM Euler Script Medium	5-10
CMCSC	CM Caps and Small Caps	8-9*	EUEX Euler Compatible Extension	7-10
CMMIB	CM Math Italic Boldface	5-9*	MSAM Symbols	5-10
CMBSY	CM Bold Symbols	5-9*	MSBM Symbols (w/Blackboard Bold)	5-10
EURB	Euler Cursive Boldface	5-10	WNCYR Cyrillic Upright	5-10**
EURM	Euler Cursive Medium	5-10	WNCYI Cyrillic Italic	5-10**
EUFB	Euler Fraktur Boldface	5-10	WNCYB Cyrillic Boldface	5-10**
EUFM	Euler Fraktur Medium	5-10	WNCYSC Cyrillic Caps and Small Caps	10**
EUSB	Euler Script Boldface	5-10	WNCYSS Cyrillic Sans Serif	8-10**

\* 10 point is included in the standard  $\text{\TeX}$  distribution.

\*\* Developed by the University of Washington

#### AMSFonts for use on a PC or mainframe

- Font Resolution: 118, 180, 240, 300, 400 dpi (one resolution per order).
- Magnification: All the standard  $\text{\TeX}$  magnifications are included. The standard magnifications are: 100, 109.5, 120, 144, 172.8, 207.4, and 248.8%.
- Format: high-density 5.25" diskettes.
- Prices: \$45 list, \$41 AMS member. A free upgrade is available (until September 1, 1990), for those who have purchased a previous version.

#### AMSFonts for use on a Macintosh with *Textures*

- Font Resolution: 72, 144, and 300 dpi (all resolutions included in each order).
- Magnification: The standard distribution includes fonts at 100% and 120%. An extended distribution, containing all the standard  $\text{\TeX}$  magsteps, is also available.
- Format: double-sided double-density 3.5" diskettes.
- Prices: *Standard* (magsteps 0-1): \$30 list, \$27 AMS member. *Extended* (magsteps 0-5): \$45 list, \$41 AMS member. A free upgrade is available (until September 1, 1990), for those who have purchased a previous version.

**SHIPPING AND HANDLING CHARGE:** \$8 per order in the US and Canada, \$15 elsewhere.

**HOW TO ORDER:** Prepayment is required. Send orders to: American Mathematical Society, P. O. Box 1571, Annex Station, Providence, RI 02901. When ordering AMSFonts for the PC, specify desired resolution. For more information: Call the AMS at (401) 455-4166, or (800) 321-4AMS in the continental U.S. and Canada, or write to:  $\text{\TeX}$  Library, American Mathematical Society, P.O. Box 6248, Providence, RI 02940.

# Do more and do it better with new PCTEX.

**PC T<sub>E</sub>X, PC T<sub>E</sub>X/386 & Big PC T<sub>E</sub>X/386, Versions 3.0**

Feature/Specification	PC T <sub>E</sub> X 2.93	PC T <sub>E</sub> X 3.0	PC T <sub>E</sub> X/386 3.0	Big PC T <sub>E</sub> X/386 3.0
<b>Page &amp; Memory Capacity</b> You won't see "T <sub>E</sub> X Capacity Exceeded"!	mem_max	65534 (1.00)	<b>131070</b> (Double!)	<b>131070</b> (Double!)
<b>Hyphenation Table Size</b> Space for hyphenation patterns	trie_size	15000 (1.00)	<b>30000</b> (Double!)	<b>60000</b> (Quadruple!)
<b>Trie Op Size</b> Complexity of hyphenation patterns	trie_op_size	255 (1.00)	<b>1024</b> (4.02)	<b>1024</b> (4.02)
<b>Maximum Trie Ops Per Language</b> Especially important for Dutch and German hyphenation		N/A	<b>512</b>	<b>512</b>
<b>Buffer Size</b> Maximum # of characters on input lines	buf_size	1024 (1.00)	<b>1500</b> (1.46)	<b>1500</b> (1.46)
<b>Stack Size</b> Maximum # of simultaneous input sources	stack_size	200 (1.00)	<b>200</b> (1.00)	<b>200</b> (1.00)
<b>Maximum # of Strings</b> Maximum # of characters in strings	max_strings	4500 (1.00)	<b>5000</b> (1.11)	<b>5000</b> (1.11)
<b>String Pool</b> Maximum # of characters in strings	pool_size	50000 (1.00)	<b>50000</b> (1.00)	<b>60000</b> (1.20)
<b>Save Size</b> Space for saving values outside current group	save_size	600 (1.00)	<b>2000</b> (3.33)	<b>2000</b> (3.33)
<b>Maximum # of T<sub>E</sub>X Commands</b>	hash_size	3000 (1.00)	<b>5000</b> (1.66)	<b>5000</b> (1.66)
<b>Minimum Free RAM Required</b>		385K (1.00)	<b>385K</b> (1.00)	<b>1.3M</b> (3.38)
<b>Minimum Free RAM Recommended</b> Memory recommended for optimum performance		550K (1.00)	<b>550K</b> (1.00)	<b>1.3M</b> (2.36)
<b>Font Memory</b> For TFM data storage	font_mem_size	51199 (1.00)	<b>65534</b> (1.28)	<b>65534</b> (1.28)
<b>Maximum Fonts Per Job</b>	font_max	127 (1.00)	<b>127</b> (1.00)	<b>127</b> (1.00)
<b>List Price</b> Order yours today!		\$249 (1.00)	<b>\$249</b> (1.00)	<b>\$295</b> (1.18)
<b>Upgrade Price</b> From PC T <sub>E</sub> X 2.93 or earlier version		\$50 (1.00)	<b>\$50</b> (1.00)	<b>\$99</b> (1.98)
				<b>\$149</b> (2.98)

## This all adds up to...

More power, greater performance, and increased memory capacity for the latest versions of popular macro packages like L<sup>A</sup>T<sub>E</sub>X and A<sub>M</sub>S-T<sub>E</sub>X. And all three new PCTEX products feature the character sets and hyphenation tables to handle even the most complex European languages.

**Order today. Call (415) 388-8853.**



12 Madrona Avenue  
Mill Valley, CA 94941

# TYPESETTING: JUST \$2.50 PER PAGE!

Send us your TeX 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 PCTEX® 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.**

---

TY  
PE  
2000

---

**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



# TEX Users

## Take Note . . .

Computer Composition Corporation offers the following services to those who are creating their technical files using **TEX**:

- Convert your DVI files to fully paginated typeset pages on our APS-5 phototypesetters at 1400 dpi resolution.
- Files can be submitted on magnetic tape or PC diskettes.
- Provide 300 dpi laser-printed page proofs which simulate the typeset page. (Optional service \$1.50 per page)
- Macro writing and keyboarding from traditionally prepared manuscripts **in several typeface families** via the **TEX** processing system. Send us your manuscript for our review and quotation.
- Full keylining and camera work services, including halftones, line art, screens and full-page negatives or positives for your printer.
- Quick turnaround (**usually less than 48 hours!**) on customer supplied DVI files of 500 typeset pages or less.
- From DVI files: first 100 typeset pages at \$4.75 per page; 100 pages and over at \$3.50 per page. **Lower prices for slower turnaround service.**

---

*For further information and / or a specific quotation,  
call or write Frank Frye or Tim Buckler*

---



**COMPUTER COMPOSITION CORPORATION**

1401 West Girard Avenue • Madison Heights, MI 48071

**(313) 545-4330    FAX (313) 544-1611**

*— Since 1970 —*

# T<sub>E</sub>X

T  
H  
E

DEPTH

A  
R  
B  
O  
R  
T  
E

DEDICATION

X  
T

DIVERSITY

W  
A  
Y

See You In T<sub>E</sub>XAS  
for the  
11th Annual TUG Meeting

535 W. William St., Ann Arbor, MI 48103, (313) 996-3566, FAX (313) 996-3573

All product names are trademarks or registered trademarks of their respective owners.

---

ArborText's T<sub>E</sub>X products reflect more than 10 year's experience and the depth of our professional development staff. Our T<sub>E</sub>X programs are flexible, fast, and loaded with features. Our work with T<sub>E</sub>X 3.0 and virtual fonts is yet another example of how our experience keeps us on the leading edge of T<sub>E</sub>X development. Be sure and see our demonstration at the TUG Conference in June.

---

ArborText continues its tradition of dedication to its customers with a knowledgeable and responsive customer support staff, the ArborText T<sub>E</sub>X Software Newsletter, and frequent software upgrades. ArborText is also a distributor of T<sub>E</sub>X support products including *Mathematica*<sup>TM</sup> and HiJaak.<sup>TM</sup>

---

ArborText offers more T<sub>E</sub>X products for more platforms than anyone. Whether you work on a Sun, Apollo, HP, DEC or PC, ArborText has the T<sub>E</sub>X solution for you. ArborText has added 5 new options since January, 1990: Introducing Preview for X11 on Sun, Apollo, HP 9000/3xx and VAX/VMS DEC Windows. Watch for Preview on DEC3100 in June!



ARBORTEXT INC.

# CAPTURE

## Put Graphics in T<sub>E</sub>X

**CAPTURE** is the graphics solution for PC-based T<sub>E</sub>X. **CAPTURE** places graphics in T<sub>E</sub>X documents produced on IBM PC systems with Hewlett-Packard LaserJet printers (and compatibles). Doesn't require PostScript.

Designed for T<sub>E</sub>X. Carefully removes all 28 LaserJet control codes that disrupt T<sub>E</sub>X. Tested with PCT<sub>E</sub>X,  $\mu$ T<sub>E</sub>X, and T<sub>E</sub>XPLUS. This ad was made using **CAPTURE** and T<sub>E</sub>XPLUS on an HP LaserJet.

"Captures" the graphics generated by any application program, including "paint" programs, circuit design, CAD, scientific data plotters, optics design, terminal emulators, clip art, spreadsheets, databases — anything that supports the LaserJet.

**Supports PostScript!** Graphics can be converted to the PK/TFM format of T<sub>E</sub>X, and used with PostScript drivers. View your graphics on screen previewers.

Graphics can be manipulated by T<sub>E</sub>X! Plots are defined as an `\hbox{}` or `\mbox{}`. Do anything with graphics you can do with type; graphics and text are handled the same.

Price: \$137.

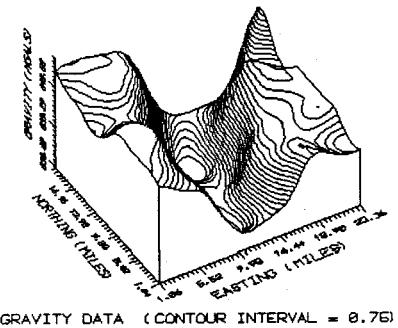
Introductory offer: \$115.

**Prime Distributor:**

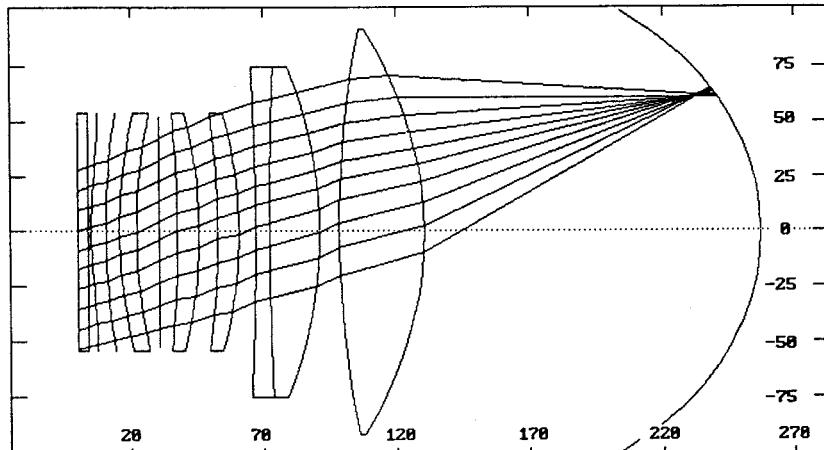
Micro Programs, Inc.  
251 Jackson Avenue  
Syosset, NY 11791  
(516) 921-1351

**Other Distributors:**

Oregon House Software  
T<sub>E</sub>X Users Group



GRAVITY DATA (CONTOUR INTERVAL = 0.76)



**Graphics Acknowledgements:** Top figure produced on PC Paintbrush, by Z-Soft, Inc. Middle figure produced on SURFER, by Golden Software, Inc. Lower figure produced on OPTEC-II/87 by SCIOPT Enterprises, Inc.

**Trademarks:** Hewlett-Packard and LaserJet are trademarks of Hewlett-Packard, Inc., IBM and PC are trademarks of International Business Machines, Inc., PostScript is a trademark of Adobe Systems, Inc., PCT<sub>E</sub>X is a trademark of Personal T<sub>E</sub>X, Inc.,  $\mu$ T<sub>E</sub>X is a trademark of ArborText, Inc., T<sub>E</sub>XPLUS is a trademark of Oregon House Software, Inc. CAPTURE is a trademark of Wynne-Manley Software, Inc. T<sub>E</sub>X is a trademark of the American Mathematical Society.

# **COMPUTERS & TYPESETTING: Errata and Changes**

As of 25 March 1990

This document contains all known errata and changes to *Computers & Typesetting*, Volumes A–E, as compiled by the TeX Project at Stanford, from 20 February 1989 through the date shown above. An up-to-date log of changes is available on-line to users with Internet access. Changes can be found in the files

- /tex/errata:errata.one (through 22 August 1984),
  - /tex/errata:errata.two (through 6 January 1986),
  - /tex/errata:errata.three (through 15 June 1987),
  - /tex/errata:errata.four (through 20 February 1989),
  - /tex/errata:errata.five (through 30 September 1989), and
  - /tex/errata:errata.tex (the most recent changes).

All errata files are on the LaBrea system at Stanford ([OLABREA.Stanford.Edu](http://OLABREA.Stanford.Edu)).

The full current errata list is published only once a year, and distributed with an appropriate issue of TUGboat. Supplements appear as necessary in subsequent TUGboat issues. The date of the last entry in each section of this document is listed in the contents, below, so that it is not necessary to check the detail to see whether anything has been added.

## CONTENTS

Distributed with TUGboat Volume 11 (1990), No. 2. Published by

T<sub>E</sub>X Users Group

P. O. Box 9506

Providence, R.I. 02940-9506, U.S.A.

### **Colophon for Computers & Typesetting**

The five volumes of *Computers & Typesetting* were composed by *TeX* (*TeX82*) using the Computer Modern (CM85) fonts as produced by *METAFONT* (*METAFONT84*). Thus, the books themselves describe exactly how they were prepared for printing. Camera-ready copy was set on an Autologic APS Micro-5 typesetter at Stanford University from font images resident on the host computer and shipped to the typesetter a character at a time, as needed.

The proof-style illustrations in Volume E were also set on the Micro-5, each figure comprising two images that were combined photographically with the text material after the images of the character shapes had been screened. The "color separation" to produce those proofs was done by a program written for that purpose.

All the copy on the cover, spine, and book jackets was also typeset by the APS, using Computer Modern fonts, except for the ISBN number.

The books were printed on Finch Opaque, basis 50 lb. acid-free paper, which has a life expectancy of several hundred years. The hardcover edition was printed and bound by Halliday Lithograph Corp., Hanover, Massachusetts, as were the spiral-bound editions of *The TeXbook* and *The METAFONTbook*.

### **Corrections to earlier editions**

Corrections and changes to the AMS/Digital Press edition of the *TeX* and *METAFONT* manual (December 1979) and to the *TeX78* and *METAFONT79* programs are described in the booklet *TeX and METAFONT: Errata and Changes* dated September 1983 (originally distributed with *TUGboat* Volume 4, No. 2). This document also contains a comparison of *TeX78* (formerly known as *TeX80*) with *TeX82*.

Errata to editions of *The TeXbook* published prior to 1986 are described in *The TeXbook: Errata and Changes* dated February 1986 (originally distributed with *TUGboat* Volume 7, No. 1).

Periodically, collections entitled *Computers & Typesetting: Errata and Changes* are compiled and distributed, containing errata as well as listings of changes to *TeX.WEB*, *MF.WEB* and to the *METAFONT* sources of the Computer Modern fonts from the documentation files *TeX82.BUG*, *MF84.BUG* and *CM85.BUG* for the corresponding period. Collections for these periods have appeared so far: through 15 June 1987; 16 June 1987 through 20 February 1989; the present collection, beginning 21 February 1989.

These documents are available from TUG; for information on how to obtain copies, write to TUG at the address on the front cover.

**Bugs in *Computers & Typesetting***

25 March 1990

This is a list of all corrections made to *Computers & Typesetting*, Volumes A–E, between 20 February 1989 and 30 September 1989 (when TeX Version 3.0 was fully defined and *The TeXbook* went into its seventeenth printing). Corrections made to the softcover version of *The TeXbook* are the same as corrections to Volume A. Corrections to the softcover version of *The METAFONTbook* are the same as corrections to Volume C. Some of these corrections have already been made in reprintings of the books. Several minor changes to Volumes A and C are not shown here because they simply make room for the more substantive changes needed to describe the new features of TeX Version 3.0 and METAFONT Version 2.0. Hundreds of changes will soon be made to Volumes B and D because of the upgrades to TeX and METAFONT; it will unfortunately be impossible to document all of those changes. Therefore, readers who need up-to-date information on the TeX and METAFONT programs should refer to the WEB source files until new printings of Volumes B and D are issued.

Corrections prior to 20 February 1989 were published by the TeX Users Group and are available from the TUG office.

---

Volume A, in general

(9/23/89)

[Change ‘127’ to ‘255’ and ‘128’ to ‘256’ in contexts referring to character codes. This happens on pages 37(twice), 39, 41, 43, 44(twice), 48, 93, 154, 277, 305(twice), 308(twice), 313, and 343. Also change ‘7-bit’ to ‘8-bit’ on pages 214 and 277.]

---

Page A23, line 16

(9/23/89)

This is TeX, Version 3.0 (preloaded format=plain 89.7.15)

---

Page A34, new copy for bottom of page

(9/23/89)

 If you use TeX format packages designed by others, your error messages may involve many inscrutable two-line levels of macro context. By setting `\errorcontextlines=0` at the beginning of your file, you can reduce the amount of information that is reported; TeX will show only the top and bottom pairs of context lines together with up to `\errorcontextlines` additional two-line items. (If anything has thereby been omitted, you'll also see ‘...’.) Chances are good that you can spot the source of an error even when most of a large context has been suppressed; if not, you can say `'I\errorcontextlines=100\oops'` and try again. (That will usually give you an undefined control sequence error and plenty of context.) Plain TeX sets `\errorcontextlines=5`.

---

Page A45, lines 9–15

(9/23/89)

`^~` has an internal code between 64 and 127, TeX subtracts 64 from the code; if the code is between 0 and 63, TeX adds 64. Hence code 127 can be typed `^~?`, and the dangerous bend sign can be obtained by saying `{\manual^~?}`. However, you must change the category code of character 127 before using it, since this character ordinarily has category 15 (invalid); say, e.g., `\catcode`^~?=12`. The `^~` notation is different from `\char`, because `^~` combinations are like single characters; for example, it would not be permissible to say `\catcode`\char127`, but `^~` symbols can even be used as letters within control words.

---

Page A45, new copy before line 20

(9/23/89)

 There's also a special convention in which `^` is followed by two "lowercase hexadecimal digits," 0–9 or a–f. With this convention, all 256 characters are obtainable in a uniform way, from `^^00` to `^^ff`. Character 127 is `^^7f`.

[Also remove one of the two dangerous bend signs on line 20.]

---

Page A45, bottom paragraph and footnote

(9/23/89)

 People who install TeX systems for use with non-American alphabets can make TeX conform to any desired standard. For example, suppose you have a Norwegian keyboard containing the letter æ, which comes in as code 241 (say). Your local format package should define `\catcode`æ=11`; then you could have control sequences like `\særttrykk`. Your TeX input files could be made readable by American installations of TeX that don't have your keyboard, by substituting `^f1` for character 241. (For example, the stated control sequence would appear as `\s^firtrykk` in the file; your American friends should also be provided with the format that you used, with its `\catcode`^f1=11`.) Of course you should also arrange your fonts so that TeX's character 241 will print as æ; and you should change TeX's hyphenation algorithm so that it will do correct Norwegian hyphenation. The main point is that such changes are not extremely difficult; nothing in the design of TeX limits it to the American alphabet. Fine printing is obtained by fine tuning to the language or languages being used.

 European languages can also be accommodated effectively with only a limited character set. For example, let's consider Norwegian again, but suppose that

[Now continue with the text on line 11 of page 46.]

---

Page A47, lines 9–21

(9/23/89)

 If TeX sees a superscript character (category 7) in any state, and if that character is followed by another identical character, and if those two equal characters are followed by a character of code  $c < 128$ , then they are deleted and 64 is added to or subtracted from the code  $c$ . (Thus, `^A` is replaced by a single character whose code is 1, etc., as explained earlier.) However, if the two superscript characters are immediately followed by two of the lowercase hexadecimal digits 0123456789abcdef, the four-character sequence is replaced by a single character having the specified hexadecimal code. The replacement is carried out also if such a trio or quartet of characters is encountered during steps (b) or (c) of the control-sequence-name scanning procedure described above. After the replacement is made, TeX begins again as if the new character had been present all the time. If a superscript character is not the first of such a trio or quartet, it is handled by the following rule.

 If TeX sees a character of categories 1, 2, 3, 4, 6, 8, 11, 12, or 13, or a character of category 7 that is not the first of a special sequence as just described, it converts the character to a token by attaching the category code, and goes into state  $M$ . This is the normal case; almost every nonblank character is handled by this rule.

---

Page A48, line 15

(9/23/89)

the input line '`$x^2$` \TeX `^62`^6`?`'

---

Page A54, third line from the bottom

(9/23/89)

For example, a well-designed TeX font for French might well treat accents as lig-

---

Page A76, lines 3–5 from the bottom

(9/23/89)

TeX does not assign any value to `\sfcode`042`.

---

Page A107, new copy for top of page (9/23/89)

 If you want to avoid overfull boxes at all costs without trying to fix them manually, you might be tempted to set `\tolerance=10000`; this allows arbitrarily bad lines to be acceptable in tough situations. But infinite tolerance is a bad idea, because TeX doesn't distinguish between terribly bad and preposterously horrible lines. Indeed, a tolerance of 10000 encourages TeX to concentrate all the badness in one place, making one truly unsightly line instead of two moderately bad ones, because a single "write-off" produces fewest total demerits according to the rules. There's a much better way to get the desired effect: TeX has a parameter called `\emergencystretch` that is added to the assumed stretchability of every line when badness and demerits are computed, in cases where overfull boxes are otherwise unavoidable. If `\emergencystretch` is positive, TeX will make a third pass over a paragraph before choosing the line breaks, when the first passes did not find a way to satisfy the `\pretolerance` and `\tolerance`. The effect of `\emergencystretch` is to scale down the badnesses so that large infinities are distinguishable from smaller ones. By setting `\emergencystretch` high enough (based on `\hsize`) you can be sure that the `\tolerance` is never exceeded; hence overfull boxes will never occur unless the line-breaking task is truly impossible.

---

Page A116, lines 11–15 (6/7/89)

 If you have two or more `\topinsert` or `\pageinsert` commands in quick succession, TeX may need to carry them over to several subsequent pages; but they will retain their relative order when they are carried over. For example, suppose you have pages that are nine inches tall, and suppose you have already specified 4 inches of text for some page, say page 25. Then suppose you make seven topinserts in a row, of

---

Page A125, lines 13–29 (9/23/89)

 When the best page break is finally chosen, TeX removes everything after the chosen breakpoint from the bottom of the "current page," and puts it all back at the top of the "recent contributions." The chosen breakpoint itself is placed at the very top of the recent contributions. If it is a penalty item, the value of the penalty is recorded in `\outputpenalty` and the penalty in the contribution list is changed to 10000; otherwise `\outputpenalty` is set to 10000. The insertions that remain on the current page are of three kinds: For each class  $n$  there are unsplit insertions, followed possibly by a single split insertion, followed possibly by others. If `\holdinginserts > 0`, all insertions remain in place (so that they might be contributed again); otherwise they are all removed from the current page list as follows: The unsplit insertions are appended to `\box n`, with no interline glue between them. (Struts should be used, as in the `\vfootnote` macro of Appendix B.) If a split insertion is present, it is effectively `\vsplit` to the size that was computed previously in Step 4; the top part is treated as an unsplit insertion, and the remainder (if any) is converted to an insertion as if it had not been split. This remainder, followed by any other floating insertions of the same class, is held over in a separate place. (They will show up on the "current page" if `\showlists` is used while an `\output` routine is active; the total number of such insertions appears in `\insertpenalties` during an `\output` routine.) Finally, the remaining items before the best break on the current page are put together in a `\vbox`

---

Page A131, line 12 (9/22/89)

work fine; but sometimes you want to have uniformity between different members of a

---

Page A155, lines 3–5 (9/23/89)

when it encounters a character that is given explicitly as `\char<number>`.

Page A214, lines 19–24

(9/23/89)

- `\the<special register>`, where `<special register>` is one of the integer quantities `\prevgraf`, `\deadcycles`, `\insertpenalties`, `\inputlineno`, `\badness`, or `\parshape` (denoting only the number of lines of `\parshape`); or one of the dimensions `\pagetotal`, `\pagegoal`, `\pagestretch`, `\pagefilstretch`, `\pagefillstretch`, `\pagefillstretch`, `\pageshrink`, `\pagedepth`. In horizontal modes you can also refer to a special integer, `\the\spacefactor`; in vertical modes there's a special dimension, `\the\prevdepth`.

Page A229, new copy after line 11

(9/23/89)

 T<sub>E</sub>X will report the badness of glue setting in a box if you ask for the numeric quantity `\badness` after making a box. For example, you might say

```
\setbox0=\linef{\trialtexta}
\ifnum\badness>250 \setbox0=\linef{\trialtextb}\fi
```

The badness is between 0 and 10000 unless the box is overfull, when `\badness=1000000`.

Page A271, lines 17–20

(9/23/89)

```
| <countdef token> | \count<8-bit number> | <codename><8-bit number>
| <chardef token> | <mathchardef token> | \parshape | \inputlineno
| \hyphenchar<font> | \skewchar<font> | \badness
```

Page A272, lines 3–4

(9/23/89)

value is between 0 and  $2^8 - 1 = 255$ ; a `<4-bit number>` is similar.

Page A273, insert after lines 11, 20, 21, 21, 38

(9/23/89)

```
\holdinginserts (positive if insertions remain dormant in output box)
\language (the current set of hyphenation rules)
\lefthyphenmin (smallest fragment at beginning of hyphenated word)
\righthyphenmin (smallest fragment at end of hyphenated word)
\errorcontextlines (maximum extra context shown when errors occur)
```

Page A274, insert after line 4

(9/23/89)

```
\emergencystretch (reduces badnesses on final pass of line-breaking)
```

Page A275, line 13

(9/23/89)

That makes a total of 103 parameters of all five kinds.

Page A283, line 14

(9/23/89)

```
| \noboundary | \unhbox | \unhcopy | \valign | \vrule
```

Page A286, lines 3–12 from the bottom

(9/23/89)

- `<letter>, <otherchar>, \char<8-bit number>, <chardef token>, \noboundary`. The most common commands of all are the character commands that tell T<sub>E</sub>X to append a character to the current horizontal list, using the current font. If two or more commands of this type occur in succession, T<sub>E</sub>X processes them all as a unit, converting to ligatures and/or inserting kerns as directed by the font information. (Ligatures and kerns may be influenced by invisible “boundary” characters at the left and right, unless `\noboundary` appears.) Each character command adjusts `\spacefactor`, using the `\sfcode` table as described in Chapter 12. In unrestricted horizontal mode, a ‘`\discretionary{}{}{}`’ item is appended after a character whose code is the `\hyphenchar` of its font, or after a ligature formed from a sequence that ends with such a character.

---

Page A287, insert after line 19 (9/23/89)

- `\setlanguage{number}`. See the conclusion of Appendix H.

---

Page A289, lines 9–14 from the bottom (9/23/89)

$2^{15} - 1$ . This is done by replacing the character number by its `\mathcode` value. If the `\mathcode` value turns out to be  $32768 = "8000$ , however, the `<character>` is replaced by an active character token having the original character code (0 to 255); TeX forgets the original `<character>` and expands this active character according to the rules of Chapter 20.

---

Page A290, insert before 13th line from bottom (9/23/89)

- `\noboundary`. This command is redundant and therefore has no effect; boundary ligatures are automatically disabled in math modes.

---

Page A296, line 16 from the bottom (9/22/89)

[There should be a ‘^’ just above the ‘3’ in the line below. This was mistakenly dropped by the printer some time during 1985; it was correct in the first two printings and it has always been correct inside the computer!]

---

Page A309, lines 3–5 (9/23/89)

**8.4.**  $\$_3 \ x_{11} \ ^7 \ 2_{12} \ \$_3 \ ^{13} \ \sqcup_{10}$  [TeX]  $b_{12} \ v_{12} \ \sqcup_{10}$ . The final space comes from the `<return>` placed at the end of the line. Code  $\wedge\wedge 6$  yields `v` only when not followed by 0–9 or `a-f`. The initial space is ignored, because state  $N$  governs the beginning of the line.

---

Page A314, line 27 (9/23/89)

The English word ‘eighteen’ might deserve similar treatment. TeX’s hyphenation algorithm will not make such spelling changes automatically.

---

Page A318, line 19 (3/3/89)

```
\def\clearnotenumber{\notenumber=0\relax}
```

---

Page A330, line 3 (8/25/89)

**20.10.** `\def\overpaid{\count0=\balance}`

---

Page A336, lines 4–8 (9/23/89)

badness rating of a box is at most 10000, except that the `\badness` of an overfull box is 1000000. INITEX initializes `\tolerance` to 10000, thereby making all line breaks feasible. Penalties of 10000 or more prohibit breaks; penalties of  $-10000$  or less make breaks mandatory. The cost of a page break is 100000, if the badness is 10000 and if the associated penalties are less than 10000 in magnitude (see Chapter 15).

Page A336, lines 2-16

(9/23/89)

ifies characters whose codes differ by 64 from the codes of ?, @, A; this convention applies only to characters with ASCII codes less than 128. There are 256 possible characters, hence 256 entries in each of the \catcode, \mathcode, \lccode, \uccode, \sfcodes, and \delcode tables. All \lccode, \uccode, and \char values must be less than 256. A font has at most 256 characters. There are 256 \box registers, 256 \count registers, 256 \dimen registers, 256 \skip registers, 256 \muskip registers, 256 \toks registers, 256 hyphenation tables. The "at size" of a font must be less than 2048 pt, i.e.,  $2^{11}$  pt. Math delimiters are encoded by multiplying the math code of the "small character" by  $2^{12}$ . The magnitude of a \dimen value must be less than 16384 pt, i.e.,  $2^{14}$  pt; similarly, the \factor in a \fil \dimen must be less than  $2^{14}$ . A \mathchar or \spacefactor or \sfcodes value must be less than  $2^{15}$ ; a \mathcode or \mag value must be less than or equal to  $2^{15}$ , and  $2^{15}$  denotes an "active" math character. There are  $2^{16}$  sp per pt. A \delcode value must be less than  $2^{24}$ ; a \delimiter, less than  $2^{27}$ . The \end command sometimes contributes a penalty of  $-2^{30}$  to the current page. A \dimen must be less than  $2^{30}$  sp in absolute value; a \number must be less than  $2^{31}$  in absolute value.

Page A348, line 12 from the bottom

(9/23/89)

\showboxbreadth=5 \showboxdepth=3 \errorcontextlines=5

Page A364, insert before line 18 from the bottom

(9/23/89)

\lefthyphenmin=2 \righthyphenmin=3 % disallow x- or -xx breaks

Page A364, line 5 from the bottom

(9/23/89)

\def\fmtname{plain}\def\fmtversion{3.0} % identifies the current format

Page A369, insert before line 5 from the bottom

(9/23/89)

Modern keyboards allow 256 codes to be input, not just 128; so TeX represents characters internally as numbers in the range 0–255 (i.e., '000–'377, or "00–"FF). Implementations of TeX differ in which characters they will accept in input files and which they will transmit to output files; these subsets can be specified independently. A completely permissive version of TeX allows full 256-character input and output; other versions might ignore all but the visible characters of ASCII; still other versions might distinguish the tab character (code '011) from a space on input, but might output each tab as a sequence of three characters ^I.

Page A370, lines 3–7

(9/23/89)

close as possible to the ASCII conventions. (b) Make sure that codes '041–'046, '060–'071, '141–'146, and '160–'171 are present and that each unrepresentable internal code < '200 leads to a representable code when '100 is added or subtracted; then all 256 codes can be input and output. (c) Cooperate with everyone else who shares the same constraints, so that you all adopt the same policy. (See Appendix J for information about the TeX Users Group.)

Page A370, bottom line

(9/23/89)

doesn't matter if these symbols have their plain TeX meanings or not. (6) There is a special convention for representing characters 0–255 in the hexadecimal forms ^00–^ff, explained in Chapter 8. This convention is always acceptable as input, when ^ is any character of catcode 7. Text output is produced with this convention only when representing characters of code  $\geq 128$  that a TeX installer has chosen not to output directly.

---

Page A385, line 8

(5/14/89)

\def\beginbox{\setbox0=\hbox\bgroup}

---

Page A400, line 18 from the bottom

(9/23/89)

page prematurely if you want to pass a signal. (Set `\holdinginserts` positive to pass a signal when the contents of `\box255` will be sent back through the page builder again, if any insertions are present.)

---

Page A419, lines 4–6

(9/23/89)

shortened or lengthened anyway; book preparation with TeX, as with type, encourages interaction between humans and machines.) The lines of the quotations are set flush right by using `\obeylines` together with a stretchable `\leftskip`:

---

Page A444, lines 21–26

(9/23/89)

following one, using the specified family and the current size, then insert the ligature character and continue as specified by the font; two characters may collapse into one, or a new character may appear. Otherwise if the font information shows a kern between the current symbol and the next, insert a kern item after the current Ord atom and move to the next item after that. Otherwise (i.e., if no ligature or kern is specified between the present text symbol and the following character), go to Rule 17.

---

Page A453, lines 12–14 from the bottom

(9/23/89)

Exception: The character ‘.’ is treated as if it were a ⟨letter⟩ of code 0 when it appears in a pattern. Code 0 (which obviously cannot match a nonzero `\lccode`) is used by TeX to represent the left or right edge of a word when it is being hyphenated.

---

Page A454, lines 7–15 from the bottom

(9/23/89)

 If a trial word  $l_1 \dots l_n$  has been found by this process, hyphenation will still be abandoned unless  $n \geq \lambda + \rho$ , where  $\lambda = \max(1, \leftlefthyphenmin$ ) and  $\rho = \max(1, \righthyphenmin)$ . (Plain TeX takes  $\lambda = 2$  and  $\rho = 3$ .) Furthermore, the items immediately following the trial word must consist of zero or more characters, ligatures, and implicit kerns, followed immediately by either glue or an explicit kern or a penalty item or a whatsit or an item of vertical mode material from `\mark`, `\insert`, or `\vadjust`. Thus, a box or rule or math formula or discretionary following too closely upon the trial word will inhibit hyphenation. (Since TeX inserts empty discretionaries after explicit hyphens, these rules imply that already-hyphenated compound words will not be further hyphenated by the algorithm.)

---

Page A455, new copy after line 13

(9/23/89)

 So far we have assumed that TeX knows only one style of hyphenation at a time; but in fact TeX can remember up to 256 distinct sets of rules, if you have enough memory in your computer. An integer parameter called `\language` selects the rules actually used; every `\hyphenation` and `\patterns` specification appends new rules to those previously given for the current value of `\language`. (If `\language` is negative or greater than 255, TeX acts as if `\language = 0`.) All `\patterns` for all languages must be given before a paragraph is typeset, if INITEX is used for typesetting.

 TeX is able to work with several languages in the same paragraph, because it operates as follows. At the beginning of a paragraph the "current language" is defined to be 0. Whenever a character is added to the current paragraph (i.e., in unrestricted horizontal mode), the current language is compared to `\language`; if they differ, the current language is reset and a whatsit node specifying the new current language is inserted before the character. Thus, if you say '`\def\french{\language1...}`' and 'mix {\french franc/ais} with English', TeX will put whatsits before the `f` and the `w`; hence it will use language 1 rules when hyphenating `franc/ais`, after which it will revert to language 0. You can insert the whatsit yourself (even in restricted horizontal mode) by saying `\setlanguage{number}`; this changes the current language but it does not change `\language`.

---

Page A459, right column	(9/23/89)
-------------------------	-----------

\*`\badness`, 214, 229, 271.

---

Page A461, right column	(9/23/89)
-------------------------	-----------

caron, see háček.

---

Page A464, line 10	(5/15/89)
--------------------	-----------

displays, 87, 103, 139–145, 166–167,

---

Page A464, right column	(9/23/89)
-------------------------	-----------

\*`\emergencystretch`, 107, 274.

---

Page A465, left column	(9/23/89)
------------------------	-----------

\*`\errorcontextlines`, 34, 273, 348.

---

Page A466, entry for 'fractions'	(9/23/89)
----------------------------------	-----------

[Add page 332 to this entry.]

---

Page A466, entry for 'French'	(9/23/89)
-------------------------------	-----------

[Add page 455 to this entry.]

---

Page A467, entry for 'hexadecimal'	(9/23/89)
------------------------------------	-----------

[Add pages 45, 47–48 to this entry.]

---

Page A467, right column	(9/23/89)
-------------------------	-----------

\*`\holdinginserts`, 125, 273, 400.

---

Page A467, bottom line	(9/23/89)
------------------------	-----------

\*`\hyphenation`, 277, 419, 452–453, 455.

---

Page A468, right column	(9/23/89)
-------------------------	-----------

infinite badness, 97, 107, 111, 229, 317.

---

Page A468, right column	(9/23/89)
-------------------------	-----------

\*`\inputlineno`, 214, 271.

---

Page A469, entry for kerns	(9/23/89)
----------------------------	-----------

[Add pages 286 and 444 to this entry.]

---

Page A469, left column	(9/23/89)
------------------------	-----------

\*\language (hyphenation method), 273, 455.

---

Page A469, right column	(9/23/89)
-------------------------	-----------

\*\lefthyphenmin, 273, 364, 454.

---

Page A470, entry for ligatures	(9/23/89)
--------------------------------	-----------

[Add pages 286 and 444 to this entry.]

---

Page A472, left column	(9/23/89)
------------------------	-----------

\*\noboundary, 283, 286, 290.

---

Page A473, right column	(9/23/89)
-------------------------	-----------

overfull boxes, 27–30, 94, 229, 238,  
302–303, 307, 400.  
avoiding, 107.

---

Page A474, left column	(9/23/89)
------------------------	-----------

\*\patterns, 277, 453, 455.

---

Page A476, left column	(9/23/89)
------------------------	-----------

\*\righthyphenmin, 273, 364, 454.

---

Page A476, right column	(9/23/89)
-------------------------	-----------

\*\setlanguage, 287, 455.

---

Page A476, right column	(9/23/89)
-------------------------	-----------

\*\showboxbreadth, 273, 302, 303, 348.  
\*\showboxdepth, 79, 273, 302, 303, 348.

---

Page A479, left column	(9/23/89)
------------------------	-----------

\*\tolerance, 29–30, 91, 94, 96, 107, 272,  
317, 333, 342, 348, 364, 451.

---

Page A481, right column, last six entries	(9/23/89)
---	-----------

$\frac{1}{2}$ , 67, 332.  
 $\frac{1}{2}$ , in unslashed form, 141, 186.  
(4-bit number), 271.  
(8-bit number), 271, 276–278.  
(15-bit number), 271, 277, 289, 291.  
(27-bit number), 271, 289, 291.

---

Page A483, lines 15 and 21	(9/23/89)
----------------------------	-----------

[Delete these two lines, as TUG's address is no longer c/o AMS.]

---

Page Bvii, top two lines (4/21/89)

*WEB documentation for four utility programs that are often used in conjunction with T<sub>E</sub>X: P<sub>OOL</sub>t<sub>E</sub>P<sub>L</sub>, T<sub>F</sub>t<sub>O</sub>P<sub>L</sub>, P<sub>L</sub>t<sub>O</sub>T<sub>F</sub>, and D<sub>V</sub>I<sub>T</sub>ype.*

---

Page B2, line 32 (6/20/89)

```
define banner ≡ 'This is TeX, Version 2.991' { printed when TEX starts }
```

---

Page B118, lines 2-4 (3/2/89)

```
begin if cur_level > level_one then
  begin check_full_save_stack; save_type(save_ptr) ← insert_token;
  save_level(save_ptr) ← level_zero; save_index(save_ptr) ← t; incr(save_ptr);
  end;
```

---

Page B182, line 13 becomes two lines (6/20/89)

*k, kk: small\_number; { number of digits in a decimal fraction }*  
*p, q: pointer; { top of decimal digit stack }*

---

Page B182, line 15 from the bottom (6/20/89)

```
begin k ← 0; p ← null; get_token; { point_token is being re-scanned }
```

---

Page B182, line 11 from the bottom (6/20/89)

```
begin q ← get_avail; link(q) ← p; info(q) ← cur_tok - zero_token; p ← q; incr(k);
```

---

Page B182, line 8 from the bottom (6/20/89)

```
done1: for kk ← k downto 1 do
  begin dig[kk - 1] ← info(p); q ← p; p ← link(p); free_avail(q);
  end;
f ← round_decimals(k);
```

---

Page B332, lines 11 and 12 from the bottom (4/8/89)

```
begin if cur_align = null then confusion('endv');
q ← link(cur_align); if q = null then confusion('endv');
```

---

Page B466, line 5 becomes three lines (6/7/89)

```
mmode + halign: if privileged then
  if cur_group = math_shift_group then init_align
  else off_save;
```

---

Page B518, line 25 (8/31/89)

```
undump(lo_mem_stat_max + 1)(lo_mem_max)(rover); p ← mem_bot; q ← rover;
```

---

Volume C, in general	(9/23/89)
----------------------	-----------

---

[Change '127' to '255' and '128' to '256' in contexts referring to character codes. This happens on pages 188(thrice) and 251.]

---

Page C91, lines 12 and 13	(8/31/89)
---------------------------	-----------

---

```
\mode=cheapo; input newface
```

and the same file should also produce a high-resolution font if we start with

---

Page C204, line 4	(8/18/89)
-------------------	-----------

---

so that *currenttransform* multiplies all *y* coordinates by *aspect\_ratio*, when paths are

---

Page C212, lines 24-27	(9/30/89)
------------------------	-----------

---

*boundarychar*      the right boundary character for ligatures and kerns

All of these quantities are numeric. They are initially zero at the start of a job, except for *year*, *month*, *day*, and *time*, which are initialized to the time the run began; furthermore, *boundarychar* is initially -1. A *granularity* of zero is equivalent to *granularity* = 1. A preloaded base file like plain METAFONT will usually give nonzero values to several other internal quantities on this list.

---

Page C259, lines 16 and 17 from the bottom	(5/14/89)
--	-----------

---

screenchars; screenstrokes; imagerules; gfcorners; nodisplays;  
notransforms; input <filename>.

---

Page C282, the three lines following the chart	(9/30/89)
--	-----------

---

METAFONT can also be configured to accept any or all of the character codes 128-255. However, METAFONT programs that make use of anything in addition to the 95 standard ASCII characters cannot be expected to run on other systems, so the use of extended character sets is discouraged.

---

Page C316, bottom 14 lines and top 30 of page C317	(9/30/89)
--	-----------

---

Ligature information and kerning information is specified in short "ligtable programs" of a particularly simple form. Here's an example that illustrates most of the features (although it is not a serious example of typographic practice):

---

```
ligitable "f": "f" =: oct"013", "i" !=: oct"020", skipto 1;
ligitable "o": "b": "p": "e" kern .5u#, "o" kern .5u#, "x" kern-.5u#,
1:: "!" kern u#;
```

---

This sequence of instructions can be paraphrased as follows:

Dear TeX, when you're typesetting an 'f' with this font, and when the following character also belongs to this font, look at it closely because you might need to do something special: If that following character is another 'f', replace the two f's by character code oct"013" [namely 'ff']; if it's an 'i', retain the 'f' but replace the 'i' by character code oct"020" [a dotless 'i']; otherwise skip down to label '1::' for further instructions. When you're typesetting an 'o' or 'b' or 'p', if the next input to TeX is 'e' or 'o', add a half unit of space between the letters; if it's an 'x', subtract a half unit; if it's an exclamation point, add a full unit. The last instruction applies also to exclamation points following 'f' (because of the label '1::').

When a character code appears in front of a colon, the colon "labels" the starting place for that character's ligature and kerning program, which continues to the end of the ligtable statement. A double colon denotes a "local label"; a `skipto` instruction advances to the next matching local label, which must appear before 128 ligtable steps intervene. The special label `||:` can be used to initiate ligtable instructions for an invisible "left boundary character" that is implicitly present just before every word; an invisible "right boundary character" equal to `boundarychar` is also implicitly present just after every word, if `boundarychar` lies between 0 and 255.

The general syntax for ligtable programs is pretty easy to guess from these examples, but we ought to exhibit it for completeness:

```

⟨ligtable command⟩ → ligtable ⟨ligtable program⟩⟨optional skip⟩
⟨ligtable program⟩ → ⟨ligtable step⟩ | ⟨ligtable program⟩ , ⟨ligtable step⟩
⟨optional skip⟩ → , , skipto ⟨code⟩ | ⟨empty⟩
⟨ligtable step⟩ → ⟨code⟩⟨ligature op⟩⟨code⟩
    | ⟨code⟩ kern ⟨numeric expression⟩
    | ⟨label⟩⟨ligtable step⟩
⟨ligature op⟩ → =: | |=: | |=:> | =:= | |=:> | |=:| | |=:> | |=:>>
⟨label⟩ → ⟨code⟩ : | ⟨code⟩ :: | ||:
⟨code⟩ → ⟨numeric expression⟩ | ⟨string expression⟩

```

A ⟨code⟩ should have a numeric value between 0 and 255, inclusive, after having been rounded to the nearest integer; or it should be a string of length 1, in which case it denotes the corresponding ASCII code (Appendix C). For example, "A" and 64.61 both specify the code value 65. Vertical bars to the left or right of '=' tell TeX to retain the original left and/or right character that invoked a ligature. Additional '>' signs tell TeX to advance its focus of attention instead of doing any further ligtable operations at the current character position.

---

Page C338, lines 21 and 22

(9/30/89)

and 127–255 have to be specified with the '#' option, on non-fancy installations of TeX, and so does code 35 (which is the ASCII code of '#' itself).

---

Page C346, left column, after line 14

(9/30/89)

\*|=:, 316, 317.  
 \*|=:>, 317.  
 \*=:, 317.  
 \*=:>, 317.  
 \*|=:, 317.  
 \*|=:>, 317.  
 \*|=:>>, 317.

---

Page C346, left column, after line 31

(9/30/89)

\*::: (local label), 317.  
 \*||: (left boundary label), 317.

---

Page C347, left column

(9/30/89)

\*`boundarychar`, 212, 317.

---

Page C352, left column

(9/30/89)

[Change '⟨ligature replacement⟩' to '⟨ligature op⟩'.]

---

Page C354, left column

(9/30/89)

⟨optional skip⟩, 217.

---

Page C356, left column

(9/30/89)

\*`skipto`, 316, 317.

---

Page Dvi, bottom two lines, and top lines of page vii (4/21/89)

■ "METAFONTware" by Donald E. Knuth, Tomas G. Rokicki, and Arthur L. Samuel, Stanford Computer Science Report 1255 (Stanford, California, April 1989), 207 pp. *The WEB programs for four utility programs that are often used in conjunction with METAFONT: GFtype, GFtoPK, GFtoDVI, and MFT.*

---

Page D63, line 9 (8/31/89)

*mem*, so we allow pointers to assume any *halfword* value. The minimum memory index represents

---

Page D63, line 28 (8/31/89)

*null = mem\_min < lo\_mem\_max < hi\_mem\_min < mem\_top ≤ mem\_end ≤ mem\_max.*

---

Page D67, in the July 1987 printing (4/7/89)

[Delete line 7, which has a redundant 'if *r* = *p* then'; move line 8 to the left 10 points for alignment; and restore the following line (which was deleted by mistake after line 8):

*node\_size(p) ← q - p { reset the size in case it grew }*

These corrections are needed only in the reprinting made July, 1987.]

---

Page D228, in the July 1987 printing (4/7/89)

[Delete lines 14–15, which were inserted erroneously from a previous errata list; and restore the following lines (which were deleted by mistake):

```

begin double(max_coef); double(x0); double(x1); double(x2);
double(y0); double(y1); double(y2);
end

```

These corrections are needed only in the reprinting made July, 1987.]

---

Page D248, in the July 1987 printing (4/7/89)

[Delete line 16, which begins with 'd ← take\_fraction'; and restore the following line (which was deleted by mistake after line 22):

*if d < alpha then d ← alpha*

These corrections are needed only in the reprinting made July, 1987.]

---

Page D389, line 10 (6/20/89)

*help1(`The expression above should have been a number >= 3/4.');*

---

Page D504, line 25 (8/31/89)

*undump(lo\_mem\_stat\_max + 1)(lo\_mem\_max)(rover); p ← mem\_min; q ← rover;*

---

Page D510, in the July 1987 printing (4/7/89)

[Move the 7th-to-last line, which begins with '*internal[fontmaking]*', one line down, and indent it to the right by 10 more points. This correction is needed only in the reprinting made July, 1987.]

---

Page Exiii, bottom four lines

(5/5/89)

- "Metamarks: Preliminary studies for a Pandora's Box of shapes" by Neenie Billawala, Stanford Computer Science Report 1256 (Stanford, California, May 1989), 132 pp. *Lavishly illustrated studies in parameter variation, leading to the design of a new family of typefaces called Pandora.*

---

Page E401, bottom line

(5/16/89)

```
math_fit( -.3cap_height# * slant - .5u#, ic#);  
penlabels(1, 2, 3, 4, 5, 6, 7, 8); endchar;
```

[some points and labels are missing at the tip of the tail on page 400]

**Bugs in Computers & Typesetting**

25 March 1990

This is a list of all corrections made to *Computers & Typesetting*, Volumes A, C, and E, since 30 September 1989 (when the revisions for *T<sub>E</sub>X* Version 3.0 and *METAFONT* Version 2.0 were made). Corrections made to the softcover version of *The T<sub>E</sub>Xbook* are the same as corrections to Volume A. Corrections to the softcover version of *The METAFONTbook* are the same as corrections to Volume C. Some of the corrections below have already been made in reprintings of the books. Hundreds of changes, too many to list here, have been made to Volumes B and D because of the upgrades to *T<sub>E</sub>X* and *METAFONT*. Readers who need up-to-date information on the *T<sub>E</sub>X* and *METAFONT* programs should refer to the WEB source files until new printings of Volumes B and D are issued.

Corrections prior to 20 February 1989 were published by the *T<sub>E</sub>X* Users Group and are available from the TUG office.

---

Page A99, line 4 from the bottom

(2/22/90)

to be chosen because there was no feasible way to keep total demerits small.

---

Page A156, line 2

(11/18/89)

Commands like `\mathchardef\alpha="010B` are used in Appendix B to define

---

Page A171, lines 24–26

(3/13/90)

formula produces a result exactly equivalent to '`\left(\langle subformula\rangle\right)`', when the `\langle subformula\rangle` doesn't end with Punct, except that the delimiters are forced to be of the `\big` size regardless of the height and depth of the subformula.

---

Page A193, lines 16–18

(12/2/89)

line if you insert '`\noalign{\break}`' after the `\cr` for that line. You can prohibit all breaks in an `\eqalignno` if you set `\interdisplaylinepenalty=10000`; or you can enclose the whole works in a `\vbox`:

---

Page A233, bottom 9 lines, and top three on next page

(12/2/89)



The `\+` macro in Appendix B works by putting the `\langle text\rangle` for each column that's followed by `\&` into an `\hbox` as follows:

```
\hbox to <column width>{\langle text\rangle\hss}
```

The `\hss` means that the text is normally flush left, and that it can extend to the right of its box. Since `\hfill` is "more infinite" than `\hss` in its ability to stretch, it has the effect of right-justifying or centering as stated above. Note that `\hfill` doesn't shrink, but `\hss` does; if the text doesn't fit in its column, it will stick out at the right. You could cancel the shrinkability of `\hss` by adding `\hfilneg`; then an oversize text would produce an overfull box. You could also center some text by putting '`\hss`' before it and just '`\&`' after it; in that case the text would be allowed to extend to the left and right of its column. The last column of a `\+` line (i.e., the column entry that is followed by `\cr`) is treated differently: The `\langle text\rangle` is simply put into an `\hbox` with its natural width.

---

Page A254, line 5 from the bottom

(10/5/89)

`\vsize` hasn't changed, and if all insertions have been held in place, the same page break

---

Page A286, lines 30-32

(3/13/90)

reading and expanding this `\par` token, TeX will see the <vertical command> token again. (The current meaning of the control sequence `\par` will be used; `\par` might no longer stand for TeX's `\par` primitive.)

---

Page A290, lines 12-13

(3/24/90)

simply a single Ord atom without subscripts or superscripts, or an Acc whose nucleus is an Ord, the enclosing braces are effectively removed.

---

Page A340, nonblank line 11

(3/13/90)

```
\topglue 1in % This makes an inch of blank space (1in=2.54cm).
```

---

Page A342, line 6

(3/13/90)

`\topglue` but not `\hglue`. It does not illustrate `\raggedright` setting of para-

---

Page A346, lines 20-21

(12/3/89)

streams used by `\read` and `\write`, to math families used by `\fam`, to sets of hyphenation rules used by `\language`, and to insertions (which require `\box`, `\count`, `\dimen`, and `\skip` registers all having the same number).

---

Page A346, line 20 from the bottom

(12/3/89)

manent value. These macros use registers `\count10` through `\count20` to hold the

---

Page A346, lines 8-13 from the bottom

(12/3/89)

number was allocated. The inside story of how allocation is actually performed should be irrelevant when the allocation macros are used at a higher level; you mustn't assume that `plain.tex` really does allocation in any particular way.

```
\count10=22 % this counter allocates \count registers 23, 24, 25, ...
```

---

Page A347, lines 2-5

(12/3/89)

```
\count19=0 % this counter allocates language codes 1, 2, 3, ...
```

```
\count20=255 % this counter allocates insertions 254, 253, 252, ...
```

```
\countdef\insc@unt=20 % nickname for the insertion counter
```

```
\countdef\allocationnumber=21 % the most recent allocation
```

```
\countdef\m@ne=22 \m@ne=-1 % a handy constant
```

---

Page A347, new line after former line 17

(12/3/89)

```
\outer\def\newlanguage{\alloc@9\language\chardef\@cclvi}
```

---

Page A352, new line before line 6 from the bottom

(3/13/90)

```
\def\topglue{\nointerlineskip \vglue-\topskip \vglue} % for top of page
```

---

Page A355, line 8 from the bottom

(12/3/89)

```
\noindent{\bf#1.\enspace}{\$1#2\par}%
```

---

Page A363, lines 8-9 from the bottom

(12/8/89)

```
\if@mid \dimen@=\ht0 \advance\dimen@ by\dp\z@ \advance\dimen@ by12\p@  
\advance\dimen@ by\pagetotal \advance\dimen@ by-\pageshrink
```

---

Page A375, line 27 (10/30/89)

depending on whether or not \t contains an asterisk. (Do you see why?) And here's

---

Page A393, lines 3-5 from the bottom (12/3/89)

```
\hskip-.17em plus-3em minus.11em
\vadjust{}\penalty10000
\leaders\copy\ hbox\hskip3.3\wd\ hbox plusifil minus.3\wd\ hbox
```

---

Page A444, line 4 (3/13/90)

Shift box *x* down by  $\frac{1}{2}(h(x) - d(x)) - a$ , where  $a = \sigma_{22}$ , so that the operator character

---

Page A450, line 8 (12/3/89)

```
0h0e0n5a0t0 1n0a0 0n2a0t0 1t0i0oo 2i0oo 0o2n0
```

---

Page A450, line 14 (12/3/89)

```
.ohoy3pohoe2n5a4t2i0o2n0.
```

---

Page A450, lines 19 and 20 (12/3/89)

```
0o2n0 0o0n1c0 1c0a0 1n0a0 0n2a0t0 1t0i0oo 2i0oo 0o2n0
```

and this yields 'o<sub>c</sub>o<sub>o</sub>n<sub>1</sub>c<sub>0</sub>a<sub>t</sub>e<sub>1</sub>n<sub>2</sub>a<sub>1</sub>t<sub>2</sub>i<sub>0</sub>o<sub>2</sub>n<sub>0</sub>' , i.e., 'con-cate-na-tion'.

---

Page A455, last lines before the quotes (11/30/89)

sit yourself (even in restricted horizontal mode) by saying \setlanguage{number}; this changes the current language but it does not change \language. Each what-sit records the current \lefthyphenmin and \righthyphenmin.

---

Page A467, right column (12/3/89)

```
*\hfilneg, 72, 100, 233, 283, 285, 290, 397.
```

---

Page A468, right column (12/2/89)

```
\interdisplaylinepenalty, 193, 349, 362.
```

---

Page A469, left column (12/3/89)

```
*\language (hyphenation method), 273, 346, 455.
```

---

Page A469, right column (10/30/89)

```
*\lefthyphenmin, 273, 364, 454, 455.
```

---

Page A472, left column (12/3/89)

```
\newlanguage, 346, 347.
```

---

Page A476, left column (10/30/89)

```
*\righthyphenmin, 273, 364, 454, 455.
```

---

Page A479, new entry (3/13/90)

```
\topglue, 340, 352.
```

---

Page A480, right column	(3/13/90)
-------------------------	-----------

---

\vglue, 352, 408.

---

Page A483, the Providence lines	(10/8/89)
---------------------------------	-----------

---

[Change the first one to

Providence RI 02940\kern.05em-9506, USA.

Then the second one will be

Providence RI 02940-9506, USA.

The second line will also appear on page C361.]

---

Page C220, top line	(3/13/90)
---------------------	-----------

---

modes you get into by hitting 'S', 'R', or 'Q', respectively, in response to error messages

---

Page C252, line 16	(3/13/90)
--------------------	-----------

---

```
for i:=1 upto n_windows: display blankpicture inwindow i; endfor
```

---

Page C264, lines 4-6 from the bottom	(3/24/90)
--------------------------------------	-----------

---

```
vardef counterclockwise primary c =
  if turningcheck>0:
    interim autorounding:=0;
    if turningnumber c <= 0: reverse fi fi c enddef;
```

---

Page C306, line 6	(3/13/90)
-------------------	-----------

---

```
ligtable "": "" =: oct"042", % close quotes
```

---

Page C309, second line from bottom	(11/18/89)
------------------------------------	------------

---

```
define_whole_vertical_blackter_pixels(vair,slab, ...);
```

---

Page C315, line 9 from the bottom	(1/2/90)
-----------------------------------	----------

---

units of printer's points):

---

Page C337, line 4 from the bottom	(1/7/90)
-----------------------------------	----------

---

```
\def\startfont{\font\testfont=\fontname \spaceskip=0pt
```

---

Page E325, line 13 (3/13/90)

if serifs:  $x_{3r} = \max(x_{1r}, \text{hround}(x_1 + .5dot\_diam - .2jut) - .5tiny)$   
else:  $x_3 = x_1 - .5 \text{ fi};$

---

Page E483, line 4 (3/13/90)

% Character codes '000-'100 and '133-'177 are generated.

---

Page E544, line 5 (3/13/90)

: (the rest of the program for 'γ' in greekl comes here)

---

Page E557, line 9 (3/13/90)

'Nevermore—Ah nevermore.' "

---

Page E558, line 21 (3/13/90)

Clasp a rare and radiant maiden whom the angels name Lenore."

---

Page E570, lines 27–28 look better with proper skewchars (3/13/90)

Here's some bold 10-point math:  $\hat{A}_0^\Gamma + \check{B}_1^\Delta - \tilde{C}_2^\Theta \times \acute{D}_3^\Lambda / \grave{E}_4^\Xi \oplus \dot{F}_5^\Pi \ominus \ddot{G}_6^\Sigma \otimes \check{H}_7^\Phi \oslash \bar{I}_8^\Psi \odot \vec{J}_9^\Omega$ .

## Changes to the Programs and Fonts

25 March 1990

The volume of changes to the **T<sub>E</sub>X** and **METAFONT** programs is too large to include in TUGboat. This material incorporates all the changes made to upgrade **T<sub>E</sub>X** to version 3.0 and **METAFONT** to version 2.0. The listing of these changes will be available from the TUG office, and inquiries should be directed there.

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

Changes numbered 351–389 are included in the updates to **T<sub>E</sub>X**, ending with the following:

-----Here I draw the line with respect to further changes

390. (I sincerely hope that there won't be any more)

### **METAFONT**

Changes numbered 547–554 are included in the **METAFONT** updates, ending with the following:

-----Here I draw the line with respect to further changes

555. (I sincerely hope that there won't be any more)

### Computer Modern fonts

Changes since 20 February 1989.

```
@x in ROMANU, the letter Q (this change simply labels point 8 on proofs)
math_fit(-.3cap_height##slant-.5u#,ic#); penlabels(1,2,3,4,5,6,7); endchar;
@y
math_fit(-.3cap_height##slant-.5u#,ic#);
penlabels(1,2,3,4,5,6,7,8); endchar;
@z
```

```
@x in ROMANL, the letter i (this change by Jonathan Kew makes the dot rounder)
if serifs: x3r=max(x1r,x1+.5(dot_diam-tiny)-.2jut) else: x3=x1-.5 fi;
@y
if serifs: x3r=max(x1r,hround(x1+.5dot_diam-.2jut)-.5tiny)
else: x3=x1-.5 fi;
@z
```

-----Here I draw the line with respect to further changes

(I sincerely hope there won't be any more!)

# **T<sub>E</sub>X Users Group Membership List — Supplement**

May 1990

This supplementary list, compiled on 21 May 1990, 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 1 March 1990. Total membership: 141 institutional members and 3,391 individuals affiliated with more than 1,500 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, Site Coordinators and members of TUG Committees	2
Addresses of TUG Members, additions and changes from 1 March 1989 through 21 May 1990	4
T <sub>E</sub> X consulting and production services for sale	9

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.

Application to mail at second-class postage rate is pending at Providence, RI and additional mailing offices. Postmaster: Send address changes to the T<sub>E</sub>X Users Group, P.O. Box 9506, Providence, RI 02940, U.S.A.

Distributed with *TUGboat* Volume 11 (1990), No. 2. Published by

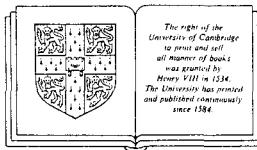
**T<sub>E</sub>X Users Group**  
P. O. Box 9506  
Providence, R.I. 02940-9506, U.S.A.

*A new and unique service from the Printing Division of the Oldest Press in the World*

# TEX-to-TYPE

The CAMBRIDGE service that lets you and your publisher decide how your mathematical or scientific text will appear.

Monotype output in Times and Helvetica as well as a complete range of Computer Modern faces from your TEX keystrokes



*For details contact*

TECHNICAL APPLICATIONS GROUP CAMBRIDGE UNIVERSITY PRESS  
UNIVERSITY PRINTING HOUSE SHAFTESBURY ROAD CAMBRIDGE CB2 2BS ENGLAND

TELEPHONE (0223) 325070



**\$79**  
IS ALL IT WILL  
TAKE TO ADD A FEW  
OF THESE TO YOUR  
TEX DOCUMENTS



With T<sub>E</sub>XPICT Graphics language\*, you will have the tools to make graphics for your T<sub>E</sub>X documents.

T<sub>E</sub>XPICT is now available from Bob Harris at:



**MICRO PROGRAMS INC**

251 Jackson Avenue, Syosset NY 11791

Telephone: (516) 921 1351.

\*TUG Boat Volume 10, No. 4, Page 627  
1989 Stanford Conference Proceedings

## Index of Advertisers

- |          |                               |          |                                |
|----------|-------------------------------|----------|--------------------------------|
| 325      | Addison-Wesley                | 320, 321 | Kinch Computer Company         |
| 316, 326 | American Mathematical Society | M-10     | Micro Programs, Inc.           |
| 331      | ArborText                     | 318      | MicroPress, Inc.               |
| cover 3  | Blue Sky Research             | 329      | Northlake Software             |
| M-10     | Cambridge University Press    | 322      | Oregon House Software          |
| 330      | Computer Composition          | 327      | Personal T <sub>E</sub> X Inc. |
| 324      | DP Services                   | 328      | Type 2000                      |
| 319, 323 | K-Talk Communications         | 332      | Wynne-Manley Software, Inc.    |