Over the years, ... the computer revolution has left its mark in a negative way: the craftsmanship that went into certain parts of composing type has been sacrificed. We're not suggesting that we go back to the old ways, but we are making a plea toward slowly raising the current standard through increasing awareness of the issues and the opportunities.

Seybold Report on Publishing Systems (August 1990)

TUGBOAT

COMMUNICATIONS OF THE TEX USERS GROUP

EDITOR BARBARA BEETON

VOLUME 11, NUMBER 4 • NOVEMBER 1990

PROVIDENCE • RHODE ISLAND • U.S.A.

TUGboat

During 1991, the communications of the TEX Users Group will be published in four issues. Two issues will consist primarily of Proceedings, one of TEX90, Cork (Vol. 12, No. 1), and the other of the 1991 TUG Annual Meeting (Vol. 12, No. 3).

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 technical items for Vol. 12, No. 2, is February 19, 1991, and for news items, March 19, 1991; the issue will be mailed in May. (Deadlines for future issues are listed in the Calendar, page 666.)

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*
Victor Eijkhout, *Associate Editor, Macros*
Jackie Damrau, *Associate Editor, LATEX*
Alan Hoenig and Mitch Pfeffer, *Associate Editors, Typesetting on Personal Computers*
See page 483 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

General Delivery

From the President

Nelson H.F. Beebe

Meetings

This has been a busy year for TUG, and for me. I've just returned from Ireland and Britain after a very successful TeX'90 meeting in Cork and a short family vacation.

The TUG'90 meeting in June in College Station is now behind us, and the *Proceedings* have appeared in record time as *TUGboat* 11, no. 3; copies were available on September 10 in Cork. I'd like to thank the authors, the program committee, the *Proceedings* editor, and the TUG office for contributing to the rapid completion of the job.

The Cork meeting was well attended, with 175 participants from 23 countries. I was pleased to have an opportunity to meet for the first time so many Europeans with whom I've exchanged letters and e-mail. Despite rain the week before and after the meeting, the weather cooperated and gave us sunshine and warmth all during the conference week. Our greatest thanks go to Peter Flynn and his support staff at Cork, the TUG office, the program committee, and all who attended for making it such an interesting meeting. Meals were served in the dining hall adjacent to the building where the lectures took place, and I think many will agree that the food was possibly the best we have enjoyed at any TUG meeting.

The weekend before the Cork meeting was devoted to a long TUG Board meeting on Saturday (which resumed again Wednesday evening), and a European summit meeting on Sunday. The Board has been dealing with difficult and divisive issues that are still not resolved; I expect to schedule a two-day board meeting at the TUG'91 meeting in Boston.

The European summit meeting was an opportunity for the heads of TUG and the European groups (five in western Europe, with five more in the early stages of formation in Czechoslovakia, Hungary, Poland, the Soviet Union, and Yugoslavia) to meet and talk about common issues. There are rumors of other regional groups forming, including one in Ireland. According to the March 1990 membership list, TUG itself has members from 48 countries.

Font standardization

One joint effort of TUG and the European groups on which excellent progress has been made is the definition of a 256-character font standard for TeX; an agreement is rapidly needed here if 8-bit fonts are to become a reality for worldwide TeX users without also becoming a terrible barrier to document portability. We hope to be able to report further on this soon (see Michael Ferguson's "Report on Multilingual Activities", p. 514, in this issue of *TUGboat*). For background, see the recent *TUGboat* articles [1, 2, 3] and references cited therein.

`tuglib` archive server

The `tuglib` server mentioned in my editorial in *TUGboat* 11, no. 1, is now fully functional at the Internet address `science.utah.edu`. A preliminary description appeared in TeXline **11**, which was distributed at the Cork meeting. At Utah, our venerable 12.5 year-old DEC-20 is slated for retirement on 31 October 1990. The name `science.utah.edu` will live on as an alias for its replacement, a UNIX machine, so as not to confuse the thousands of people who have used it. Because this will change some of the details of `tuglib` access, I decided to delay an in-depth article about `tuglib` until the next issue of *TUGboat*, when the changeover will be behind us.

Bibliography archive

Earlier this year, I began a bibliography project which is now well underway. Its eventual goal is to provide public access to a collection of BibTeX-format bibliographies of

- publications about TeX (files `texbook1.*`)
- books and journals that use TeX for their production (files `texbook2.*`)
- journals that accept articles written in TeX (files `texbook3.*`)
- literate programming
- PostScript
- digital typography

as well as to collect bibliographies for many scientific journals in fields that I'm interested in. However, I am also willing to deposit there contributed bibliographies for journals in any field of science or engineering.

An e-mail message to

`tuglib@science.utah.edu`

with the text `send index from tex/bib` will return an index of current holdings. It is intended that these holdings will be available on computer-readable magnetic media for users who lack e-mail access.

A bibliography consists of at least two files: (1) a LaTeX file that prints the entire bibliography, together with a title page, a version date and number, and a short prefacing text; and (2) one or more BibTeX `.bib` files that hold the actual entries. The bibliography produced this way is in two-column format with alphanumeric tags. This choice was intentional: it produces narrow columns that stress TeX's formatting abilities, and it detects certain errors in author names (such as an incorrectly placed *Jr.*) that would not be caught if numeric citations were used. Explicit hyphenations are supplied to reduce, if not completely eliminate, overfull boxes. The `\emergencystretch` feature of TeX 3.0 is most helpful in reducing the rivers of white that otherwise tend to occur with narrow columns; this support is hidden in an option file, `bibmods.sty`, that works with pre-3.0 versions of TeX as well.

The LaTeX files all use `showtags.sty`, a style file that I wrote for the project; it produces a boldface copy of each BibTeX citation tag in a right-adjusted framed box over the corresponding entry, which makes a typeset copy handy for reference. For another recent approach to the bibliography lookup problem, see [4].

The bibliography entries for books contain ISBN values, and for journals, ISSN values, where these are available, and modified BibTeX style files support the printing of these fields. ISBNs contain a check digit verifiable by a GNU Emacs editor function I wrote; this Emacs function is also available from `tuglib` as the file `isbn.el`. Does anyone know if the 8-digit ISSNs have a check digit, and if so, what the algorithm for computing it is?

All files in the collection contain a special comment header whose style we hope to popularize for TeXware. Here is an example, somewhat reformatted to accommodate the narrow columns of *TUGboat*:

```
%% @LaTeXstylefile{
%%     author   = "Nelson H. F. Beebe",
%%     version  = "1.01",
%%     date     = "11 Jul 1990",
%%     filename = "showtags.sty",
%%     address  = "Center for Scientific
%%                    Computing
%%                 Department of Mathematics
%%                 South Physics Building
%%                 University of Utah
%%                 Salt Lake City, UT 84112
%%                 USA
%%                 Tel: (801) 581-5254",
%%     checksum = "70     333    3033",
%%     email    = "beebe@science.utah.edu",
%%     codetable = "ISO/ASCII",
%%     keywords  = "bibtex, cite tag, latex",
%%     supported = "yes",
%%     docstring = "This style file causes
%%                  the bibliography cite
%%                  tags to be displayed in
%%                  boldface text in a
%%                  right-adjusted framed
%%                  box over each entry in a
%%                  bibliography.  This
%%                  serves as a handy
%%                  reference when the tags
%%                  are needed for a \cite{}
%%                  macro.
%%
%%                  For flexibility, the
%%                  user may redefine
%%                  \thecitetag to change
%%                  the format. E.g.
%%                  \renewcommand{\thecitetag}
%%                  [#1]{\fbox{\small\tt #1}}
%%                  would typeset the tag in
%%                  small typewriter text in
%%                  a box.
%%
%%                  The checksum field above
%%                  contains the standard
%%                  UNIX wc (word count)
%%                  utility output of lines,
%%                  words, and characters;
%%                  eventually, a better
%%                  checksum scheme should
%%                  be developed."
%%     }
```

The format is similar to that used in BibTeX files, although it is not expected to be processed by BibTeX; doing so would require writing a special `.bst` style file, and augmenting BibTeX with a simple filter to delete comment markers prefixing each line. It is not hard to keep a template for this comment header available for insertion into files that you write; the GNU Emacs file `texfile.el`, available with the collection, can be customized for your personal use to insert one.

The essential features of this comment header are:

- The `keyword = "value"` format is extensible; the ones shown above are recommended for a start, but others may be desirable in the future.
- The original filename is included in case the file itself is renamed due to constraints of some file systems.
- Author name and address, and whether the file is maintained and supported, are recorded.
- The last modification date, and major and minor version numbers, are provided. The minor

number is incremented whenever any change is made to the file.

- A documentation string is provided to hold a short abstract describing the file. The intent of the `keywords` and `docstring` fields is to provide information that can be automatically extracted for publication in a local file guide.
- The character set used for the file is recorded in the `codetable` value; this will become increasingly important as 8-bit character sets become more common and files are exchanged electronically.
- The `checksum` field can be used to detect corruption or modification of the file. Corruption of electronic mail through certain antisocial gateway machines is a regrettable fact of life, but it can also happen through file transfers between unlike systems, or more rarely, from media errors.

The current checksum scheme is too simple; it records only counts of characters, words, and lines. This does not detect errors of transposition or substitution; the latter are typical of e-mail corruption.

A better checksum system is needed, probably one based on cyclic redundancy checksums, such as the widely-used CRC-16. For our purposes, such a checksum should be *independent* of the line terminators used (CR, CR LF, LF, or other), but should still incorporate the line count; that way, the same checksum will be obtained on different file systems. Also, while the checksum is embedded in the file itself, it should not affect the checksum computation; otherwise, you could never set it correctly. Do we have volunteers for a `WEB` implementation? I can provide simple code for a fast compact implementation of the CRC-16 checksum that can be used for a start.

Other archive sites are welcome to pick up the current bibliography collection from Utah; they are warned, however, that these files are undergoing rapid evolution, and often change several times a week.

I invite you to support this project; after getting a copy of any of the bibliographies, send me corrections and additions, the latter preferably in BIBTEX format, including ISBN or ISSN fields. The scope of literature that I'm trying to cover is far larger than any single individual can manage. The `texbook2.bib` and `texbook3.bib` files in particular need to be greatly expanded.

If you have other bibliographies in BIBTEX format that you would be willing to contribute to the collection, I would like to hear about them.

TUGboat publication

TUGboat has finally reached a stage where we have a backlog of papers to publish. This introduces a publication delay, and also means that we need to modify procedures somewhat. In the past, our capable editors have undertaken the job of referee as well as the normal work of editor.

Traditionally, scientific publishing has used the referee process to improve the quality of published papers, to catch errors before they appear in print, to discourage the publication of substandard work, and to encourage concise presentation. Unfortunately, the pressure on authors to publish has resulted in a proliferation of journals whose page limits often force omission of important details; conciseness should not be achieved at the loss of readability and usability.

I hereby call for qualified volunteers to act as referees for TUGboat articles. You may send a statement to that effect to the *TUGboat* editors; it will be helpful to them if you list the subject areas in which you are willing to referee papers. In the interests of preserving rapid publication, you should be willing to carry out your referee job within a few days of receipt of a paper, and as is traditional, you will remain anonymous (except to the editors), and be under obligation not to disclose, or make use of, refereed papers before they are published.

References

[1] Nelson H. F. Beebe. Character set encoding. *TUGboat*, 11(2):171–175, June 1990.

[2] Janusz S. Bień. On standards for computer modern font extensions. *TUGboat*, 11(2):175–183, June 1990.

[3] Yannis Haralambous. TEX and latin alphabet languages. *TUGboat*, 10(3):342–345, November 1989.

[4] Michael A. Harrison and Ethan V. Munson. On integrated bibliography processing. *Electronic Publishing—Origination, Dissemination, and Design*, 2(4):193–209, December 1989.

◇ Nelson H.F. Beebe
Center for Scientific Computing
Department of Mathematics
220 South Physics Building
University of Utah
Salt Lake City, UT 84112
USA
Tel: (801) 581-5254
FAX: (801) 581-4148
Internet: `Beebe@science.utah.edu`

The Future of TEX

At the Texas and Cork conferences, birds-of-a-feather (BOF) sessions were held to discuss the future of TEX. In particular, the attendees were concerned about the maintenance of TEX when Prof. Knuth no longer wished to be involved, and about the possible evolution of incompatible TEX-like products should this occur. These discussions gave rise to a number of questions, which were presented to the TUG Board during the Cork meeting for formal response. The questions are shown here, and were also communicated to Prof. Knuth by one of the BOF attendees. Prof. Knuth has been kind enough to provide a definitive response as to the future of TEX and of METAFONT; this response is given on the following pages, along with a commentary by the TUG President, Nelson Beebe.

1. Does the TUG Board acknowledge the need to maintain and develop TEX when Prof. Knuth decides to be no longer involved?
2. Does the Board agree that TUG should oversee and coordinate changes to TEX in an attempt to improve the program as well as to establish a single standard?
3. What mechanism can be established to enable TUG to maintain control over TEX without stifling further necessary development?
4. Does the board agree that fundamental research into unsolved typographical problems is necessary to improve TEX and that TUG should promote and seek funding for such research?

Future of TEX BOF
Cork, 12 Sept 1990

Attendees of the "Future of TEX" Birds-of-a-Feather Session at TEX90, Cork:

Johannes Braams
PTT Research, The Netherlands

Tim Bradshaw
University of Edinburgh, UK

Adrian F. Clark
University of Essex, UK

Christine Detig
Germany

Angus Duggan
University of Edinburgh, UK

Victor Eijkhout
University of Nijmegen, The Netherlands

Jeremy Gibbons
Oxford University, UK

Michel Goossens
CERN, Switzerland

Klaus Guntermann
Technische Hochschule Darmstadt, Germany

Amy Hendrickson
TEXnology, Inc, USA

Alan Jeffrey
Oxford University, UK

Frank Mittelbach
Electronic Data Systems, Germany

Timothy Murphy
Trinity College Dublin, Ireland

Marion Neubauer
Universität Heidelberg, Germany

David Osborne
Nottingham University, UK

Nico Poppelier
Elsevier Science Publishers BV, The Netherlands

Thomas Reid
Texas A&M University, USA

David Rhead
Nottingham University, UK

Chris Rowley
Open University, UK

Jan Michael Rynning
Royal Institute of Technology, Sweden

Jens Schmidt
Universität Hamburg, Germany

Rainer Schöpf
Universität Heidelberg, Germany

Joachim Schrod
Germany

Alan Wittbecker
Digital Equipment Corporation, USA

Ralph Youngen
American Mathematical Society, USA

The Future of TEX and METAFONT

Donald E. Knuth

My work on developing TEX, METAFONT, and Computer Modern has come to an end. I will make no further changes except to correct extremely serious bugs.

I have put these systems into the public domain so that people everywhere can use the ideas freely if they wish. I have also spent thousands of hours trying to ensure that the systems produce essentially identical results on all computers. I strongly believe that an unchanging system has great value, even though it is axiomatic that any complex system can be improved. Therefore I believe that it is unwise to make further "improvements" to the systems called TEX and METAFONT. Let us regard these systems as fixed points, which should give the same results 100 years from now that they produce today.

The current version number for TEX is 3.1, and for METAFONT it is 2.7. If corrections are necessary, the next versions of TEX will be 3.14, then 3.141, then 3.1415, ..., converging to the ratio of a circle's circumference to its diameter; for METAFONT the sequence will be 2.71, 2.718, ..., converging to the base of natural logarithms. I intend to be fully responsible for all changes to these systems for the rest of my life. I will periodically study reports of apparent bugs, and I will decide whether changes need to be made. Rewards will be paid to the first finders of any true bugs, at my discretion, but I can no longer afford to double the size of the reward each year. Whenever I have created a new version, I will put it in the official master TEX archive, which currently resides at Stanford University. At the time of my death, it is my intention that the then-current versions of TEX and METAFONT be forever left unchanged, except that the final version numbers to be reported in the "banner" lines of the programs should become

```
TeX, Version $\pi$
```

and

```
METAFONT, Version $e$
```

respectively. From that moment on, all "bugs" will be permanent "features."

As stated on the copyright pages of Volumes B, D, and E, anybody can make use of my programs in whatever way they wish, as long as they do not use the names TEX, METAFONT, or Computer Modern. In particular, any person or group who wants to produce a program superior to mine is free to do so. However, nobody is allowed to call a system TEX or METAFONT unless that system conforms 100% to my own programs, as I have specified in the manuals for the `TRIP` and `TRAP` tests. And nobody is allowed to use the names of the Computer Modern fonts in Volume E for any fonts that do not produce identical `tfm` files. This prohibition applies to all people or machines, whether appointed by TUG or by any other organization. I do not intend to delegate the responsibility for maintenance of TEX, METAFONT, or Computer Modern to anybody else, ever.

Of course I do not claim to have found the best solution to every problem. I simply claim that it is a great advantage to have a fixed point as a building block. Improved macro packages can be added on the input side; improved device drivers can be added on the output side. I welcome continued research that will lead to alternative systems that can typeset documents better than TEX is able to do. But the authors of such systems must think of another name.

That is all I ask, after devoting a substantial portion of my life to the creation of these systems and making them available to everybody in the world. I sincerely hope that the members of TUG will help me to enforce these wishes, by putting severe pressure on any person or group who produces any incompatible system and calls it TEX or METAFONT or Computer Modern—no matter how slight the incompatibility might seem.

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

Comments on the Future of TEX and METAFONT

Nelson H.F. Beebe

1 Introduction

Donald E. Knuth's article above, "The Future of TEX and METAFONT", clearly states the Grand Wizard's wishes about these programs and the Computer Modern font family.

Where does that leave TUG? The opening paragraph of TUG's bylaws includes this statement (the emphasis is mine):

> ...specifically to identify, develop, operate, fund, support, promote and encourage charitable, educational and scientific programs and projects which will stimulate those who have an interest in *systems for typesetting technical text and font design*; to exchange information of same and associated use of computer peripheral equipment; to establish channels to facilitate the exchange of macro packages, etc., through publications and otherwise; and to develop, implement and sponsor educational programs, seminars and conferences in connection with the foregoing...

I believe that this expressly says that TUG's purview legitimately goes beyond TEX, METAFONT, and Computer Modern, whose further development has been frozen by their author in the interests of providing a constant solid base for their users, and of returning to his own extensive research and writing efforts, which have been outstanding landmarks in the development of the fields of Computer Science and Applied Mathematics.

2 TEX is international

As the TEX-related portion of the Utah bibliography project described in my President's message in this issue of *TUGboat* will attest, the use of TEX is widespread. Many books and journals are routinely typeset by TEX, including almost all of the publications of the American Mathematical Society, one of the world's largest publishers of mathematical material. Large on-line data bases in TEX input form now exist.

I suggest that no other typesetting system, or desk-top publishing system, has been used for as many languages as TEX has. TEX is in use for all major European languages, plus Arabic, Chinese, Coptic (Ethiopian), Hebrew, several Indian languages, Japanese, Persian, Russian, Thai, Turkish, Vietnamese, and likely others that I may be unaware of. This list includes languages that are written horizontally and vertically. TEX can support typesetting of multiple languages in the same text, thanks to the work of Frank Liang on hyphenation [11], of Michael Ferguson on multi-lingual TeX [4, 5, 6, 7], and of Donald Knuth and Pierre MacKay on TEX-XET [9].

These research efforts led to several features incorporated in TEX 3.0 to make multilingual typesetting standardly available. For related work in other typesetting systems, see [2] on tri-directional typesetting, and articles in the July 1987, August 1988, and May 1990 issues of the Communications of the ACM.

There are textbooks about TEX in at least Danish, Dutch, English, French, German, and Japanese, and I know of in-progress translations to Persian of the TEXbook and the LATEX User's Guide and Reference Manual.

There are TUG members in nearly 50 countries, and I'm sure there are TEX users in many more. Besides TUG, there are five thriving regional groups in Western Europe, and five or more others are forming.

3 The challenge from desk-top publishing systems

The international use of TEX suggests that Donald Knuth's decision to freeze further development will in some ways be highly beneficial. However, it does *not* imply that TEX, METAFONT, and Computer Modern are the last word in computer-based typesetting. If TUG does not pursue further development of typesetting software, TEX may be doomed to extinction far sooner than it should, for several reasons:

- Desk-top publishing is *big* business, with several tens of millions of installed personal computers forming the potential market base. The Salt Lake Tribune on 10 October 1990 carried an article on Utahns included in the just-released Forbes list of the 400 wealthiest people in the world. The two developers of Word Perfect, one of the most popular word processing systems available on personal computers, workstations, and some mainframes, have a combined worth of nearly one (North American) billion dollars; the young chairman of Microsoft Corporation is worth even more.
- Software is a commodity that is relatively cheap to produce and distribute. The actual development costs of most commercial software are only a small fraction of potential sales revenues, and the computing industry has numerous examples of the quick attainment of fab-

ulous wealth. What *does* cost a lot of money is sales and marketing, and the on-going support of software, including personnel, authoring, and documentation. This situation encourages competitiveness and rapid development of new products.

- Desk-top publishing (WYSIWYG)[1] systems are attractive to many people, particularly novices, because of the immediate feedback that they provide. With most of them, it is impossible to generate syntax errors of the type that TEX is perhaps infamous for, because input is checked character by character as it is entered, and formatting commands are generated by function keys and menu selections, rather than as embedded markup. Few of these systems today are suited to the batch typesetting required in journal and periodical production, because they bind a graphical input and output interface too tightly to the typesetting machinery; however, that market, because of its publishing volume, will eventually prove attractive.
- Users of most WYSIWYG systems are encouraged by the immediate feedback of the typeset display to make visual, rather than logical, design decisions. Design professionals often criticize visual design [10, Section 1.4] because it can lead to poor typography. Also, the visual layout may make it difficult to re-use the text, or to reformat it for a different output style. These objections may disappear as newer generations of these systems provide better support for document styles, and separation of the jobs of authoring or document entry, and document design.
- Several desk-top publishing systems are already capable of easily handling multi-column output, multi-column floats, flowing of typeset text around inserts (both rectangular and non-rectangular), and easy integration of graphics with text; these are areas where TEX is noticeably deficient.

4 TEX's advantages

In view of the points raised in the preceding section, we must then ask what does TEX (and I mean also METAFONT, Computer Modern, and related software) offer that competing desk-top publishing systems do not, at least not yet?

[1] WYSIWYG = What You See Is What You Get, sometimes called What You See Is All You've Got.

- TEX provides public-domain access to the source code of its related software. Source code of commercial implementations remains proprietary, but the changes from the public domain versions are usually in system-dependent areas that do not affect the overall operation of the software, and for most machines, both public domain and commercial implementations are available.

  Public access to the source code is extremely important. It permits both low-cost, or even free, public-domain implementations, and supported commercial implementations, of TEX to be available on many different platforms. A commercial user of TEX need not be tied to any single vendor of the software; such ties can become a significant competitive disadvantage when the supplier does not keep up with technological progress. As one such example, I cite the TV Guide experience [1].

  Although TEX is probably one of the most bug-free software packages of its size, it is reassuring to a user to know that if a question ever arises as to why the system typeset text in a particular way, the availability of well-documented source code makes it in principle possible to find the reason. Public access to source code means that bugs are often found and reported by several users, and fixes can come more quickly. By contrast, commercial desk-top publishing systems are almost always unfathomable black boxes whose surprises are indecipherable; it may be difficult to convince a vendor that an anomaly is a 'bug' instead of a 'feature'.
- TEX source code is written in a relatively portable language, and consequently, it is available *today* for virtually every commercially-available computing system, from personal computers, up to supercomputers.
- The wide availability and use, and the frozen development, of TEX mean that we can view it as an *archival* document formatting system. Most commercial publishing products have completely ignored this issue; succeeding product generations offer new features and bug fixes, but are often incompatible with earlier ones. It is certainly true that much of what is published today is "throw-away" material, and in such cases, whether the publishing system can reformat the same document years from now is of no concern.

  However, in academic circles, this is decidedly not the case. Academicians research and write in the interest of wide dissemination of their ideas, both to current colleagues, and to future generations. Authors and publishers of such material are interested in re-using it for multiple documents. One of the TEX90 speakers from a major publisher noted that in some fields of study, the same text can be re-used more than a dozen times.

- TEX's freedom from architectural and commercial licensing restrictions facilitates collaborative efforts of several authors to work on the same document, even if they have different computer hardware.
- TEX's markup is visible, not hidden in magical undocumented binary data embedded in the document. This has several virtues:
  - Detection and correction of formatting errors is usually easier when the formatting commands can be seen.
  - It is relatively easy to write simple filters that strip the markup from a document to produce raw text which is input to other software tools for spell checking, grammatical analysis, and so on.
  - The markup is recorded in the same character set as the raw text, greatly facilitating document exchange between unlike systems, or via electronic mail.
- TEX's support for visible markup means that translation may be possible between it and other markup systems, such as SGML-based ones.
- TEX supports a powerful macro language that permits the creation of separate input interfaces that can be quite different from `plain` TEX. AMS-TEX and LATEX are the most obvious examples, but the Free Software Foundation's TEXinfo and LATEXinfo systems, and the use of TEX as the typesetting engine for documents written in other markup languages, as is done at at least two major publishing houses, are other examples. Most desk-top publishing systems lack this extensibility.
- TEX is capable of handling multi-lingual typesetting; few commercial publishing systems today can make this claim.
- TEX's mathematical typesetting abilities are still unmatched by most desk-top publishing systems. Its Computer Modern font family, together with the AMS font extensions, provides a repertoire of characters that is far more comprehensive than almost anything available on other systems. (I was able to announce at the Cork meeting that Adobe Systems has finally released a Lucida font in POSTSCRIPT format with a set of mathematics characters matching Computer Modern. Lucida is the font used in the typesetting of *Scientific American.*) The public-domain nature of TEX will of course make it possible for commercial systems to incorporate TEX's sophisticated algorithms for mathematics; however, this is likely to happen slowly because most of the commercial desk-top publishing market has little need for mathematical typesetting.
- TEX, and other systems based on visible markup (including those that use SGML), have a significant advantage over WYSIWYG systems in that style and content can be clearly separated. In most desk-top publishing systems, style and content are inextricably entwined. This has important ramifications for alternate uses of the input text, for user training, and for the effort needed to change the style without modifying the content.

  With TEX, authors and clerical staff need learn only *one* system that can be used with very minor changes to produce documents in a wide variety of styles.

5 Some observations

TEX currently has a portability advantage over most other typesetting systems. Many commercial publishing products are tied very closely to the hardware or window system architecture of a specific machine, particularly in the personal computer market. This has meant years of delay in getting them ported to other systems. The rise of the C language, particularly during the 1980s, as an efficient, but nevertheless portable, machine-independent implementation language is slowly beginning to be recognized by vendors. Assembly-language coded systems are now being rewritten in C or C++ to reach a wider market. Recent examples include SAS, Word Perfect, and Lotus 1-2-3. Because of the spread of popular window systems, such as X, Microsoft Windows, and others, and the efforts to standardize them, I expect that by the end of this decade, most commercial software products related to publishing will be available on as wide a range of machines as TEX currently is.

While it is true that standard TEX does not provide an immediate visual display of the typeset text, the Berkeley VORTEX project, about which too little has been written, and ArborText's Publisher system are demonstrations that TEX *can* have such an interface. The rapid advances in computer speeds that have occurred, largely through RISC processor developments, and the volume production economizations possible through sales of millions of personal computers, suggest that we are only a few short years away from instantaneous typeset on-line display.

Few existing systems, including WYSIWYG ones and TEX, are suitable for newspaper publishing, which is characterized by its complicated layout of text and graphics in up to six or eight columns, and daily deadlines that cannot be missed without serious economic impact. I expect that the most printing done in the world today is in newspapers. While most of larger newspapers now use computer-based typesetting, I suspect that their systems are rather specialized for that industry.

6 Necessary future developments

The preceding sections have discussed the relative strengths and weaknesses of TEX versus desk-top publishing systems. I have found in discussions with other TUG members at meetings, and in mail exchanges, that many of us share the view that development of TEX cannot stand still. Donald Knuth has placed understandable restrictions on the use of the names TEX, METAFONT, and Computer Modern. Consequently, evolutionary systems arising from TEX will have to use different names.

I believe strongly that what needs to be done now is for those users of TEX and METAFONT who have pushed the limits of those systems to begin writing down detailed descriptions of just what those limitations are, and to make well thought-out suggestions about the directions that future work ought to take.

I made a start last year on the relation of TEX and graphics in [3].

Frank Mittelbach gave a wonderfully incisive exposition on the future of TEX at the College Station TUG'90 meeting [12], and followed that at the Cork TEX'90 conference with a fine presentation of work done together with Reinhard Wonneberger on the future of BIBTEX [14].

Michael Vulis has shown with an actual implementation [13] how scalable fonts tightly integrated into a TEX-like system can offer new and interesting capabilities. To those who would quibble with his incorporation of the name TEX, I would observe that VTEX is a superset of TEX, and with a special command-line argument, it will disable all extensions and perform exactly like TEX; nevertheless, it would be advisable to adhere to the Grand Wizard's wishes, and change the name.

John Hobby presented some very promising work at the Stanford TUG'89 meeting on extensions of METAFONT for generation of POSTSCRIPT output [8], and related work by Shimon Yanai and Daniel Berry should soon appear in *TUGboat*.

We need more such articles! Please, if you can contribute new ideas, and I know from personal contacts that many of you can, write them down (or even up) for publication in *TUGboat* or other journals in the field.

Only when we have a solid base of written contributions from the TEX experts will it be possible for some future researcher to have a reliable starting point for the design of the evolution of TEX to the next generation of typesetting system, and that person will have the added challenge of finding new names!

Let us hope that a major design goal of such an effort will be the maintenance of compatibility with existing TEX and METAFONT input, so that the substantial, and growing, base of existing TEX and METAFONT material will continue to be processable, with exactly the same results, by the next generation of computer-based typesetting systems. I believe that this would be far preferable to having separate, but mutually incompatible, systems that must try to coexist peacefully.

Incompatibility may eventually become necessary. By the time that TEX's grandchildren are born, it may be that they will bear little resemblance to their ancestor. We can only hope that use of TEX will have become so commercially important that translators of TEX documents to the new generation systems will be developed. An analogy can be found in programming languages: Fortran is a distant ancestor of the Algol family of languages, including Pascal, C, C++, and Ada. An enormous body of important Fortran code exists that cannot possibly be rewritten by hand; public-domain and commercial translators have been developed to convert Fortran code to some of these languages.

While the design of TEX's children is underway, we need to get all TEX systems upgraded to the final versions that Donald Knuth has provided, and we need to agree upon a standard 8-bit TEX font encoding that will permit the exchange of documents that make use of the new features of TEX 3.0. As I noted in my President's message in this issue, this second problem should soon be solved.

References

[1] Elizabeth Barnhart. TEX in the commercial environment — multi-column output. *TUGboat*, 8(2):185, July 1987.

[2] Zeev Becker and Daniel Berry. `triroff`, an adaptation of the device-independent `troff` for formatting tri-directional text. *Electronic Publishing—Origination, Dissemination, and Design*, 2(3):119–142, October 1989.

[3] Nelson H. F. Beebe. TEX and Graphics: The State of the Problem. *Cahiers GUTenberg*, 1(2):13–53, 1989. Presented to: Congrès GUTenberg, Paris, France, 16–17 May 1989.

[4] Michael Ferguson. Multilingual TEX update. *TUGboat*, 7(1):16, March 1986.

[5] Michael Ferguson. A (hopefully) final extension of multilingual TEX. *TUGboat*, 8(2):102, July 1987.

[6] Michael Ferguson. Coordination of non-English use of TEX. *TUGboat*, 11(1):8–9, April 1990.

[7] Michael J. Ferguson. A multilingual TEX. *TUGboat*, 6(2):57, July 1985.

[8] John D. Hobby. A METAFONT-like System with PostScript Output. *TUGboat*, 10(4):505–512, December 1989.

[9] Donald Knuth and Pierre MacKay. Mixing right-to-left texts with left-to-right texts. *TUGboat*, 8(1):14, April 1987.

[10] Leslie Lamport. *LATEX—A Document Preparation System—User's Guide and Reference Manual.* Addison-Wesley, 1985.

[11] Franklin Mark Liang. *Word Hy-phen-ation by Com-pu-ter.* PhD thesis, Stanford University, August 1983.

[12] Frank Mittelbach. E-TEX: Guidelines for future TEX extensions. *TUGboat*, 11(3):337–345, September 1990.

[13] Michael Vulis. VTEX extensions to the TEX language. *TUGboat*, 11(3):429–434, September 1990.

[14] Reinhard Wonneberger and Frank Mittelbach. BIBTEX reconsidered. *TUGboat*, 12(1), January 1991 (to appear).

◇ Nelson H.F. Beebe
Center for Scientific Computing
Department of Mathematics
220 South Physics Building
University of Utah
Salt Lake City, UT 84112
USA
Tel: (801) 581-5254
FAX: (801) 581-4148
Internet: `Beebe@science.utah.edu`

Editorial Comments

Barbara Beeton

This year's "meeting season"

We have come to the end of the TEX summer meeting season, and it was a busy one. I attended the TUG annual meeting in College Station, Texas, TEX90 in Cork (the 5th TEX meeting in Europe, and the first co-sponsored by TUG), and the NTG SGML-TEX Conference in Groningen. As always, one of the best parts of these conferences was the chance to greet in person all sorts of people who I'd already "met" by e-mail. Meetings in Europe are a bit less hurried than those in the U.S. — there is often time to linger over coffee before returning to the next session. But I was sorry at the Dutch meeting to miss several talks that I would like to have attended; there were two tracks, and one had to make choices.

What were the highlights? This is at least partly subjective, but I think there were some features that really stood out.

For those on the TUG Board of Directors, it was novel and welcome to be able to mingle with everyone else at lunch time at the annual meeting, and I felt that the networking lunches (an innovation this year) were a success and should become a permanent part of the meeting planning.

In both Texas and Ireland, one of the most active discussion topics was, whither TEX? We have an answer from Don Knuth concerning the software that goes by the *name* TEX (see his article on page 489, and Nelson Beebe's comments, page 490). However, I don't assume that means there can be no growth, only that it should be well-planned, concentrated in the areas of pre- and post-processing, and we should start thinking of good names.

TUG's eleventh annual meeting. The meeting at Texas A & M University wasn't attended by as many people as the Tenth at Stanford, but there was a good program for those who did come. Several papers stood out for me:

• Frank Mittelbach on what's still missing from TEX. His paper in the *Proceedings* not only explains but also illustrates the points he made.

• Helen Gibson on how an in-house system for producing high-quality exhibition catalogs and research publications with "problem scripts" (Sanskrit and South Asian) was built without alienating either the researchers or the secretaries.

• Mimi Lafrenz on running a small company that provides TEX services to publishers. Her message was that sharing information is essential to producing a quality product. She's also a dynamite speaker.

And there was a lot more; you can read it in the *Proceedings*.

The *Proceedings*, published as *TUGboat* 11, no. 3, were ready in time to be handed out in Cork. That issue had its own editor, Lincoln Durst; the timeliness and consistency of the published work is due, more than I can say, to his energetic oversight. I was permitted to relax and observe from a distance. To Lincoln and everyone else who contributed to its success, good job!

NTG's SGML & TEX Conference. The Dutch SGML Users' Group co-sponsored this conference, and with the exception of two talks on more general topics, the program proceeded along two tracks, one mainly SGML and the other mainly TEX. This would seem to have been the ideal opportunity to indulge in some cross-fertilization, but I thought that most attendees stayed with the track that was most familiar. I attended the meeting in Groningen because Kees van der Laan invited me to give a talk on the production history of *TUGboat*. Here are my impressions of some of the talks.

• Joop van Gent described the "two faces of text": the logical content of a scientific document, and the ancillary things that it is "about" — the author, the state of knowledge when the work was written, etc. He then presented some TEXniques for processing documents to make them available in a form suitable for advanced document retrieval systems.

• Malcolm Clark took a look at the problem of exchanging (TEX) documents through electronic mail. The present networks don't support 8-bit transmission, and even 7-bit is shaky at times. So what is needed is another piece for the toolkit — a filter for 8 → 7bit transmission, to give us "safe TEX".

• Sake Hogeveen's paper on aspects of scientific publishing was on the SGML track, but it managed to include two introductory courses, "TEX in five minutes" and "LATEX in six minutes", that got the idea across quite well to anyone who had never heard those names. The main points of this talk were that an "ordinary" author needs all the help he can get to make manuscript preparation simpler, and that good typography supports the structure of a document, so that a reader understands the structure better because of the typography.

• Victor Eijkhout considered the need for a metaformat which would permit the document style designer to specify attributes in a non-TEX syntax. This is important because document markup that identifies common structural elements can permit a common input view but different outputs, and the common input view permits a good typist to go at top speed. The challenge: to implement a programming language suitable for the designer's specifications.

• Johannes Braams gave an overview of the work of NTG Working Group 13, which has developed various LATEX styles appropriate for Dutch documents, and of his own work on Babel, a style for support of multilingual variation (an article describing this work will appear in *TUGboat* 12, no. 2).

Unfortunately, there will be no formal proceedings of this conference. I hope that the ideas discussed there will find their way into print in other ways, and not remain inaccessible.

TEX90 in Cork. Peter Flynn made sure that everyone felt welcome at the same institution which had been home to George Boole — University College Cork. The meeting was attended by about 175 TEXies from 23 different countries — a larger and more varied group than in Texas. Again, just a few highlights:

• Christine Detig on hypertext. I want it! I need it to organize the *TUGboat* archives!

• Malcolm Clark on why anyone who wants to edit and produce the proceedings of a TEX conference (or any similar collection) should think twice.

• Yannis Haralambous on typesetting Old German. He proposed that there is more enjoyment in reading old texts when the typefaces are appropriate, and to illustrate his point he has created METAFONT fonts for Fraktur, Schwabacher, Gotisch, and a beautiful set of ornamented capitals. This paper was deservedly awarded the prize for best presentation. (See also Yannis' paper on Arabic, page 520 in this issue.)

• Nico Poppelier on the use of SGML as a common coding system for organizing and re-using texts and data by a publisher who employs not only TEX but several other typesetting systems.

• Angela Barden on how to write a useful book on TEX. (Actually, I missed the presentation, but have heard such good things about this paper that I can hardly wait for the *Proceedings*.)

- Konrad Neuwirth on why it *isn't* suitable to teach TeX in schools. Though still a student, Konrad admits to being "not typical", and his arguments were well thought out and to the point.

The *Proceedings* will constitute *TUGboat* 12, no. 1, which will appear early next year.

TUGboat news

With this issue, there is a new Associate Editor for Macros, Victor Eijkhout. Until recently an inhabitant of Nijmegen in The Netherlands where he contributed to NTG Working Group 13 and, via the networks, to discussions on TeXhax, UKTeX, etc., Victor is now working in the Center for Supercomputing Research and Development at the University of Illinois in Urbana. We intend to keep him busy.

Two issues of *Proceedings* will be published in 1991. It has already been mentioned that the *Proceedings* of TeX90, Cork, will be *TUGboat* 12, no. 1. This issue will be on a schedule separate from that of the regular issues, as will the *Proceedings* of TUG91, which will appear as 12, no. 3.

There will be a total of four issues in 1991, the same as in 1990. Issues 2 and 4 will be regular issues. See below for a discussion of this decision.

To accommodate the expected volume of technical articles, we are prepared to increase the size of the two regular issues as much as necessary or possible, subject to financial considerations. News items, reports, and the calendar will be published in all issues. The deadline for such items for *TUGboat* 12, no. 1 will be past by the time you read this; deadlines for later issues will appear in the calendar.

The first *regular* issue will be moved to May 1991, and the editorial deadline adjusted. Now that there is a Macro Editor, we intend to pay more attention to details. We are also attempting to have technical articles refereed, to try to assure an even, high level of quality. For this, we need more time. Therefore, for regular issues there will now be two deadlines: one for technical articles, and one for news items and reports. For *TUGboat* 12, no. 2, the deadline for technical articles will be February 19, and for news items and reports, March 19.

We would like to recognize the fine work of our printer, Waverly Press, whose expeditious service in 1990 has drastically shortened the time between deadline and publication. The availability of the Texas *Proceedings* in Cork could not have happened without their cooperation.

The dues increase and number of issues

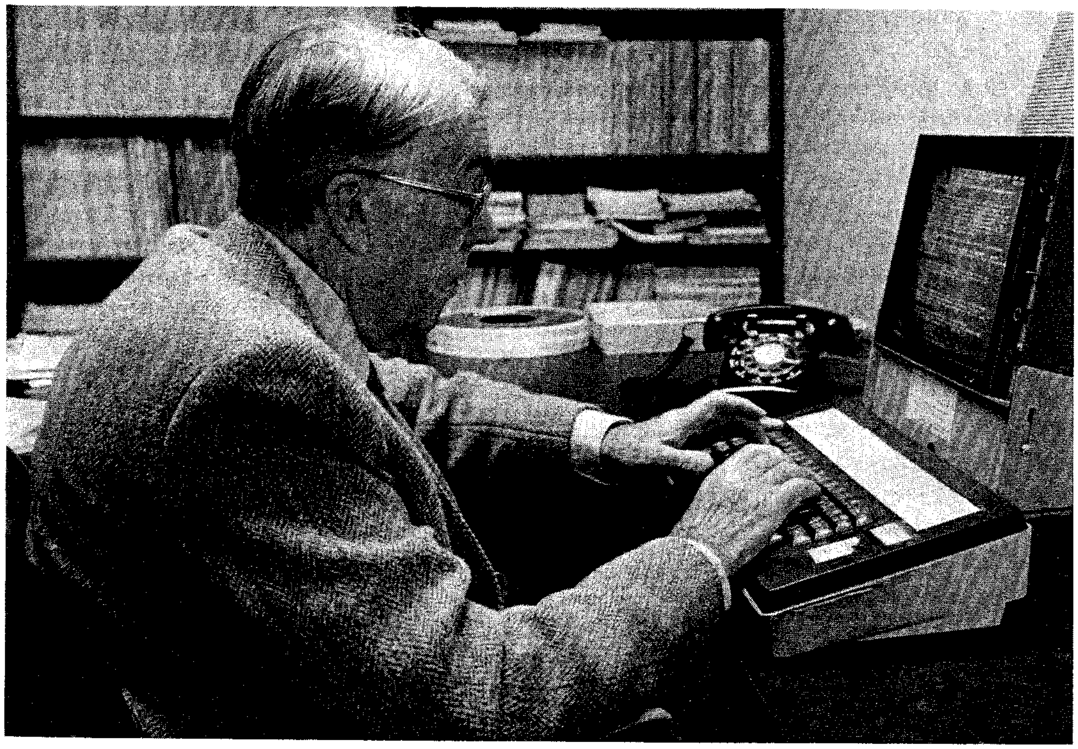
You will notice on the inside front cover and on your renewal notices that the membership dues will be $45 next year. There will be no discount for early renewal, and no postal supplements for members outside the U.S. The increase from the 1990 rates is intended to help alleviate TUG's present financial difficulties.

The TUG Board voted (at a meeting in Cork) to rescind the postal surcharge. TUG was not meant to be a U.S.-only organization, and the required (rather than optional) supplement added to the perception of members outside the U.S. that they were being treated as "second class citizens". The other problem for members outside North America, that of slow delivery (in spite of higher charges to provide air mail service) is being attacked separately, with the help of our printer; an investigation into the possibility of printing part of the edition in Europe has been deferred until there is experience with the new mailing strategy.

After discussion of whether there should be four or five issues, we have decided on the smaller number for the following reasons: (1) To add a fifth issue would have necessitated an increase in dues larger than the one approved. (2) TUG's financial situation hasn't been good for the past two years, and we're trying to attack the problem from both ends: increased revenue (membership dues) and decreased costs (holding the line at four issues, even though only two of those will be regular issues). (3) The postal permit requires that the Post Office be notified in advance of changes in the number of issues scheduled; changes are possible, but (as it has already been announced that GUTenberg will publish the TeX91 *Proceedings*; see page 667) a reversing change would be likely next year.

Regardless of these changes, we intend to do our best to continue to produce a *TUGboat* that we can be proud of, and that you will find interesting and useful.

Arthur Lee Samuel, 1901–1990



The TeX community lost its beloved senior member on July 29, when Art Samuel died at age 88. He had devoted a great deal of time during the past decade giving personal attention to the needs of thousands of TeX users all over the world.

Art joined Stanford's TeX project in the spring of 1981, and he was a major participant in all of our activities during those crucial days of the early 80s until the project completed its work in 1985. He continued to answer numerous queries about TeX, electronic and otherwise, for several more years, even though the onset of disease made it difficult for him to walk and eventually confined him to a wheelchair. He was more than 50 years older than nearly everyone else in the TeX group, but he always was a lively contributor to our meetings and a source of inspiration.

TeX was, of course, only a footnote to his long and illustrious career. He did pioneering work on vacuum tubes at Bell Labs during the 30s; he played a leading role in the development of the ILLIAC computer as a faculty member at the University of Illinois in the 40s; he directed IBM's Poughkeepsie laboratory where significant research on transistors was carried out in the 50s; he devoted considerable time to government service, for example as chair of the Joint Services Committee on Electron Devices for 17 years; he received more than 40 patents for various inventions. When David Fuchs and I traveled with him to the Cincinnati TUG meeting in January, 1982, he regaled us with interesting stories about his experiences as a pilot. He is best known for the seminal research he did on machine learning, beginning in 1949; his famous program for playing checkers won a fine game against America's number-four-ranked player in 1962 (see [3]).

I think he also had a keen interest in publishing throughout much of his life. For example, he was editor of the *IBM Journal of Research and Development* from January 1962 to July 1966. He retired from IBM in 1966 and became a member of Stanford's Artificial Intelligence Project. MIT was also wooing him at that time, hoping that he would move to Massachusetts and participate in research on publishing automation, sponsored by the American Newspaper Publishers Association. We can only guess what the history of computer typography would have become if he and his wife had not preferred to settle on the West Coast.

TeX users know Art Samuel best from his classic booklet *First Grade TeX* [6], which came out almost simultaneously with *The TeXbook*. This book was his own idea; he worked on it constantly for more than a year as TeX82 was taking shape. Naturally it was an immediate success. According to reports in *TUGboat*, more than 600 copies were sold by the TUG office in 1984; more than 800 copies in 1985; more than 1200 in 1986 [9]. And TUG was only one of several outlets for his book.

Art had long been interested in writing tutorials for beginners. For example, he had written the lead article for the IRE's first special issue on computers in 1953, entitled "Computing bit by bit, or Digital computers made easy" [2]. Every user of Stanford's SAIL computer was helped by his booklets "Essential E" [4] and "Short WAITS" [5], which provided brief introductions to the text editor and operating system. So he decided to do a similar thing for TeX. (He told me that his first goal was to write a kindergarten primer; but after awhile he found that TeX was too complicated, so he needed to go up to the first grade level.) He tried valiantly to keep the entire document at most 32 pages long. Finally, however, he accepted the 34-page length that seemed to be necessary. We can imagine his surprise when the Japanese translation was published in 1989—his book now ran to 175 pages! [8]

When I taught a special course about METAFONT in 1984, using a new operating system and a new text editor on new workstations, Art saved the day by writing key documentation so that our novice computer users could cope with the experimental equipment. He also attended the course: Some of his homework is displayed in [1].

Incidentally, he had written an article in 1964 predicting what computers would be like in 1984 [7]; people tell me that this article, by a senior researcher at IBM, was the first public prediction that personal computers would become commonplace before long. Some of his futuristic ideas of 1964 were indeed prophetic. But it is amusing to compare the orderly transition to a high-tech world envisioned in [7] with what Art himself was doing in 1984.

Surely Art must hold the all-time world record so far for correct computer instructions written after the age of 80. He did a great deal of significant programming for the TeX project, especially of device drivers; without them, we wouldn't have been able to print the results of our basic experiments when TeX82 and METAFONT84 were being created. Ultimately I printed tens of thousands of pages with his software, running it on three quite different machines. He incorporated some clever ideas about caching font data so that memory requirements would be low.

He also took a look at my GFtoDVI program, which originally had a fancy algorithm for positioning labels near the points on METAFONT proofsheets. I was quite happy with the algorithm, but he didn't like the way the labels looked in his own experiments. So he made his own personal GFtoDVI and hesitantly showed it to me. At first I thought it was terrible—my "elegant" data structure for non-overlapping rectangles had been replaced by a brute force search—but soon I had to admit that (a) Art's method gave better results than mine, and (b) it also ran faster. Needless to say, I soon abandoned my original approach and adopted his scheme. Sophisticated computer science can sometimes be too seductive.

All of us can surely be grateful for the many things Art Samuel accomplished during his lifetime, and for the many lessons he taught us. The fruits of his work will live on.

– Donald Knuth

References

[1] Donald E. Knuth, "A course on METAFONT programming," *TUGboat* **5**, 2 (November 1984), 105–118; see especially page 114.

[2] Arthur L. Samuel, "Computing bit by bit, or Digital computers made easy," *Proceedings of the Institute of Radio Engineers* **41** (October 1953), 1223–1230.

[3] Arthur L. Samuel, "Some studies in machine learning using the game of Checkers," *IBM Journal of Research and Development* **3** (1959), 210–229. Reprinted with an additional annotated game in *Computers and Thought*, edited by Edward A. Feigenbaum and Julian Feldman (New York: McGraw-Hill, 1963), 71–105.

[4] Arthur L. Samuel, *Essential E.* Stanford Computer Science Report STAN-CS-80-796 (March 1980).

[5] Arthur L. Samuel, *Short WAITS.* Stanford Computer Science Report STAN-CS-81-839 (February 1981).

[6] Arthur L. Samuel, *First Grade TeX: Beginner's TeX Manual.* Stanford Computer Science Report STAN-CS-83-985 (November 1983).

[7] Arthur L. Samuel, "Computers in 1984: The banishment of paper-work," *New Scientist* **21**, no. 380 (February 1964), 529–530.

[8] Arthur L. Samuel, *First Grade TeX*; a Japanese translation with notes (Tokyo: Kinokuniya, 1989).

[9] Samuel B. Whidden, "Acknowledgment of contributors," *TUGboat* **6**, 1 (March 1985), 5; **7**, 1 (March 1986), 7; **8**, 1 (April 1987), 5.

Software

Answers to *Exercises for TEX: The Program*

Donald E. Knuth

Editor's note: The exercises apropos to these answers were printed in *TUGboat* 11, no. 2, pp. 165–170.

1. According to the index, *initialize* is declared in §4. It is preceded there by ⟨Global variables 13⟩, and §13 tells us that the final global variable appears in §1345. Turning to §1345, we find '*write_loc*: *pointer*;' and a comment. The comment doesn't get into the Pascal code. The mini-index at the bottom of page 535 tells us that '*pointer*' is a macro defined in §115. Our quest is nearly over, since §115 says that *pointer* expands to *halfword*, which is part of the Pascal program. Page ix tells us that lowercase letters of a WEB program become uppercase in the corresponding Pascal code; page x tells us that the underline in '*write_loc*' is discarded. Therefore we conclude that 'PROCEDURE INITIALIZE' is immediately preceded in the Pascal program by 'WRITELOC:HALFWORD;'.

But this isn't quite correct! The book doesn't tell the whole story. If we actually run TANGLE on TEX.WEB (without a change file), we find that 'PROCEDURE INITIALIZE' is actually preceded by

```
{1345:}WRITELOC:HALFWORD;{:1345}
```

because TANGLE inserts comments to show the origin of each block of code.

2. The index tells us that *done5* and *done6* are never used. (They are included only for people who have to make system-dependent changes and/or extensions.)

3. Here we change the *input_ln* procedure of §31. One way is to replace the statements '$buffer[last] \leftarrow xord[f\uparrow]$; $get(f)$' by the following:

```
if ord(f↑) = '33 then
  begin get(f);
  if (ord(f↑) ≥ "@") ∧ (ord(f↑) ≤ "_") then
    begin buffer[last] ← xord[chr(ord(f↑) − '100)]; get(f);
    end
  else buffer[last] ← invalid_code;
  end
else begin buffer[last] ← xord[f↑]; get(f);
  end;
```

4. The new string essentially substitutes "quarters" q (of value 25) for "dimes" x (of value 10). Playing through the code of §69 tells us that 69 is now represented by lvvviv and 9999 is mmmmmmmmmcmqcvqiv. (The first nine m's make 9000; then cm makes 900; then qc makes 75; then vq makes 20; and iv makes the remaining 4.)

5. Because it may be decreased by 1 in §1293 before being increased by 1 in §82. (The code in §1293 decreases *error_count* because "showing" uses the *error* subroutine although it isn't really an error.)

6. The q becomes Q in §83. This causes §86 to print 'OK, entering \batchmode', after which *selector* is decreased so that '...' and ⟨return⟩ are *not* printed on the terminal! (They appear only in the log file, if it has been opened.) This is TEX's way of confirming that \batchmode has indeed been entered.

7. (a) Arithmetic overflow might occur when computing $t*297$, because $7230585 \times 297 = 2^{31}+97$. (b) Some sort of test is need to avoid division by zero when $0 < s < 297$. If $s < 1663497$ then s **div** $297 < 5601$, and 7230585/5600 is a bit larger than 1291 so we will have $r > 1290$ in such a case. The threshold value has therefore been chosen to save division whenever possible. (One student suggested that the statement '$r \leftarrow t$' be replaced by '$r \leftarrow 1291$'. That might or might not be faster, depending on the computer and the Pascal compiler. In machine language one would '**goto**' the statement that sets $badness \leftarrow inf\_bad$, but that is inadmissible Pascal.) (c) If we get to §128 with $r = p + 1$, we will try to make a node of size 1, but then there's no room for the *node_size* field. (d) If we get to §129 with only one node available, we'll lose everything and *rover* will be invalid. (Older versions of TEX have a more complicated test in §127, which would suppress going to §129 if there were two nodes available. That was unnecessarily cautious.) (e) This is a subtle one. The lower part of memory must not be allowed to grow so large that a *node_size* value could ever exceed *max_halfword* when nodes are being merged together in §127.

8. We assume that *min_quarterword* = *min_halfword* = 0.

100:	0	0	0	*type* (*hlist_node*), , *link*
101:			6553600	*width* (100 pt)
102:			0	*depth*
103:			655360	*height* (10 pt)
104:			0	*shift_amount*
105:	1	2	200	*glue_sign* (*stretching*), *glue_order* (*fill*), *list_ptr*
106:		10.0		*glue_set* (type *real*)
200:	7	1	10003	*type* (*disc_node*), *replace_count*, *link*
201:		300	10000	*pre_break*, *post_break*
300:	11	1	0	*type* (*kern_node*), *subtype* (*explicit*), *link*
301:			655360	*width* (10 pt)
400:	6	0	0	*type* (*ligature_node*), , *link*
401:	1	11	10001	*font*, *character*, *lig_ptr*
500:	12	0	600	*type* (*penalty_node*), , *link*
501:			5000	*penalty*
600:	10	0	700	*type* (*glue_node*), *subtype* (*normal*), *link*
601:		8	0	*glue_ptr* (*fill_glue*), *leader_ptr*
700:	1	0	0	*type* (*vlist_node*), , *link*
701:			655360	*width* (10 pt)
702:			32768	*depth* (0.5 pt)
703:			327680	*height* (5 pt)
704:			-327680	*shift_amount* (−5 pt)
705:	0	0	800	*glue_sign* (*normal*), *glue_order* (*normal*), *list_ptr*
706:		0.0		*glue_set* (type *real*)
800:	0	0	900	*type* (*hlist_node*), , *link*
801:			655360	*width* (10 pt)
802:			0	*depth*
803:			327680	*height* (5 pt)
804:			0	*shift_amount*
805:	0	0	10004	*glue_sign* (*normal*), *glue_order* (*normal*), *list_ptr*
806:		0.0		*glue_set* (type *real*)
900:	2	0	0	*type* (*rule_node*), , *link*
901:			-1073741824	*width* (*null_flag*)
902:			0	*depth*
903:			32768	*height* (0.5 pt)
10000:	1	"U"	400	*font*, *character*, *link*
10001:	1	"f"	10002	*font*, *character*, *link*
10002:	1	"f"	0	*font*, *character*, *link*
10003:	1	"!"	500	*font*, *character*, *link*
10004:	2	"d"	10005	*font*, *character*, *link*
10005:	2	"a"	0	*font*, *character*, *link*

9. (Norwegian Americans will recognize this as an 'Uff da' joke.) The output of *short_display* is

```
\large Uff []
```

since *short_display* shows the pre-break and post-break parts of a discretionary (but not the replacement text). However, if this box were output by *hlist_out*, the discretionary break would not be effective; the result would be a box 100 pt wide, beginning with a large '!' and ending with a small 'da', the latter being raised 5 pt and underlined with a 0.5 pt-rule.

10. Since *prev_depth* is initially *ignore_depth*, we get

```
### vertical mode entered at line 1
                                (\output routine)
prevdepth -999.99998, prevgraf 1 line
```

11. According to §236, *int_base* + 17 is where *mag* is stored. (One of the definitions suppressed by an ellipsis on page 101 is *mag*; you can verify this by checking the index!) The initial value of *mag* is set in §240. Hence *show_eqtb* branches to §242 and prints '`\mag=1000`'.

12. In the following chart, '(3)' means a value at level three, and '—' is a level boundary:

```
                                                                                       (2)
                                                                                        9
                                                                                   —    —
                                                             (1)  (1)       (1)  (1)  (1)  (1)  (1)
                                                              6    6         8    8    8    8    8
                                                   —    —    —    —
                                              (1)  (1)  (1)  (1)  (1)  (1)  (1)  (1)  (1)  (1)  (1)
                                               4    4    4    4    4    4    4    4    4    4    4
                                    (1)  (1)  (1)  (1)  (1)  (1)  (1)  (1)  (1)  (1)  (1)  (1)  (1)
                                     2    2    2    2    2    2    2    2    2    2    2    2    2
     save_stack:                   —    —    —    —    —    —    —    —    —    —    —    —    —    —
   xeq_level[p]: (1)  (1)  (1)  (1)  (2)  (1)  (2)  (2)  (1)  (3)  (1)  (1)  (2)  (2)  (3)  (2)  (2)  (1)
    eqtb[p].int:  0    1    2    2    3    4    5    5    6    7    8    8    9    9   10    9   10    8
     operations: \day=0 \g   \a   {    \a   \g   \a   {    \g   \a   \g   }    \a   {    \a   }    \a   }
```

The final value is therefore \day=8.

13. (reference count), *match* `!`, *match* `#`, *left_brace* `[`, *end_match*, *left_brace* `{`, *mac_param* `#`, *right_brace* `]`, *mac_param* `!`, *out_param* 2, *left_brace* `[`. Notice that the *left_brace* before the *end_match* is repeated at the end of the replacement text, because it has been matched (and therefore removed from the input).

14. According to §233, *show_eqtb*(*every_par_loc*) calls *show_token_list* with the limit $l = 32$. According to §292, we want the token list to contain a token that prints as many characters as possible when *tally* = 31; the value of *tally* is increased on every call to *print_char* (§58). By studying the cases in §294, we conclude that the worst case occurs when a *mac_param* is printed, and when the character *c* actually prints as three characters. The statement '*print_esc*("`ETC.`")' in §292 will print seven additional characters if the current *escape_char* is another tripler. (Longer examples are possible only if TEX has a bug that tweaks one of the outputs '`\CLOBBERED.`' or '`\BAD.`' in §293; but this can't happen.)

In other words, a worst-case example such as

```
\escapechar=`\^^M \catcode`\^^I=6
\everypar{1234567890123456789012345678901^^Ietc.}
```

in connection with the suggested test line will print

```
{restoring ^^Meverypar=1234567890123456789012345678901^^I^^I^^METC.}
```

thereby proving that 44 characters can be printed by *show_eqtb*(*every_par_loc*).

15. Here we must look at the *get_next* procedure, which scans the *buffer* in strange ways when two identical characters of category 7 (*sup_mark*) are found. After the `\catcode` of open-quote has been set to 7, *get_next* begins to scan a control sequence in §354, which goes to §355 and finds a space after `‘‘`. Since a space is code *'40*, it is changed to *'140*, and the buffer contents are shifted left 2. By strange coincidence, *'140* is again an open-quote character, so we get back to §355, which changes `‘‘(` to `h` and goes back to *start_cs* a third time. Now we go to §356 and then back to §355 and *start_cs*, having changed `‘‘)` to `i`. The fourth round, similarly, changes `‘‘‘` to a blank space, and the fifth round finishes the control sequence.

If we try to input the stated line, `INITEX` will come to a halt as follows:

```
! Undefined control sequence.
<*> \catcode‘‘=7 \hi
                    !\error
```

This proves that the *buffer* now says `\hi !`.

16. The error message in question is

```
! Undefined control sequence.
<*> \endlinechar=‘! \error
                          ^^M
```

and our job is to explain the appearance of `^^M`. The standard `\endlinechar` is *carriage_return*, according to §240; this is *'15* according to §22, and *'15* is `^^M` in ASCII code. Thus, a *carriage_return* is normally placed at the end of each line when it's read into the *buffer* (see §360). This *carriage_return* is not usually printed in an error message, because it equals the *end_line_char* (see §318). We see it now because *end_line_char* has changed.

Incidentally, if the input line had been

```
\endlinechar=‘!\error
```

(without the space after the `!`), we wouldn't have seen the `^^M`. Why not? Because TeX calls *get_next* when looking for the optional space after the ASCII constant `‘!` (see §442–443), hence the undefined control sequence `\error` is encountered before *end_line_char* has been changed!

17. One problem is to figure out which control sequence is undefined; it seems to be the '?', since this character has been made active. One clue is to observe from §312 and §314 that '`<recently read>`' can be printed only when *base_ptr* = *input_ptr*, *state* = *token_list*, *token_type* = *backed_up*, and *loc* = *null*. A token list of type *backed_up* usually contains only a single item; in that case, the control sequence name must be '`How did this happen?`', and we have a problem getting an active character into a control sequence name.

But an arbitrarily long token list of type *backed_up* can be created with the `\lowercase` operation (see §1288). In that case, however, the right brace that closes `\lowercase` is almost always still present in TeX's input state, and it would show up on the error message. (The *back_list* procedure of §323 does not clear a completed token list off of the stack.) We have to make TeX clear off its stack before the } is scanned.

At this point the exercise begins to resemble "retrograde chess" problems. Here is one solution; since it requires a very long input line, it has been broken into a three-line answer:

```
\def\answer{\let~\expandafter\lccode‘!=‘H% [line has been broken]
 ~\lowercase~{~!~o~w~ ~d~i~d~ ~t~h~i~s~  % [line has been broken]
 ~h~a~p~p~e~n~?}}
```

(The '`H`' is a lowercase '`!`'; a chain of `\expandafter`s is used to make the right brace disappear from the stack.)

Another approach uses `\csname`, and manufactures a `?` from a `!`:

```
\def\answer{\def\a##1{{\global\let##1?\aftergroup##1}}% [broken]
 \escapechar‘H\lccode‘!|/‘?                           % [broken]
 \lowercase{\expandafter\a\csname ow did this happen!\endcsname}}
```

But there is a (devious) one-line solution, which makes the invisible *carriage_return* following `\answer` into a right brace:

```
\def\answer{\catcode13=2\lccode`!=H\lowercase\bgroup!ow did this happen?}
```

18. (The answer to this problem was much more difficult to explain in class than I had thought it would be, so I guess it was also much more difficult for the students to solve than I had thought it would be. After my first attempt to explain the answer, I decided to make up a special version of TEX that would help to clarify the scanning routines. This special program, called DemoTEX, is just like ordinary TEX except that if `\tracingstats>2` the user is able to watch TEX's syntax routines in slow motion. The changes that convert TEX to DemoTEX are explained in the appendix below. Given DemoTEX, we tried a lot of simple examples of things like '`\hfuzz=1.5pt`' and '`\catcode`a=11`' before plunging into exercise 18 in which everything happens at once. While we were discussing input stacks, by the way, we found it helpful to consider the behavior of TEX on the following input:

```
\output{\botmark}
\def\a{\error}
\mark{
 \everyvbox{
  \everypar{
   \everydisplay{
    \everyhbox{
     \everymath{\noexpand\a}
     $\relax}
    \hbox\bgroup\relax}
   $$\relax}
  \noindent\relax}
 \vbox\bgroup\relax}
\hbox{}\vfill\penalty-10000
```

Here `\penalty` triggers `\botmark`, which defines `\everyvbox` and begins a `\vbox`, which defines `\everypar` and begins a `\par`, which defines `\everydisplay` and begins a `\display`, etc.)

The first line is essentially

```
\gdef\a#1d#2#3{#2}
```

where the second 'd' has catcode 12 (*other_char*). Hence the second `d` will match a `d` that is generated by `\romannumeral`. In this line, *scan_int* is called only to scan the `` `d `` and the `12`.

The second line calls *scan_dimen* in order to evaluate the right-hand side of the assignment to `\hfuzz`. After *scan_dimen* has used *scan_int* to read the '`100`', it calls *scan_keyword* in order to figure out the units. But before the units are known to be '`pt`' or '`pc`', an `\ifdim` must be expanded. Here we need to call *scan_dimen* recursively, twice; it finds the value 12 pt on the left-hand side, and is interrupted again while *scan_keyword* is trying to figure out the units on the right-hand side. Now a chain of `\expandafter`s causes `\romannumeral888` to be expanded into `dccclxxxviii`, and then we have to parse `\a dccclxxxviii`. Here `#1` will be `\else`, `#2` and `#3` will each be `c`; the expansion therefore reduces to `cclxxxviii\relax\fi`. The first '`c`' completes the second '`Pc`', and the `\ifdim` test is true. Therefore the second '`c`' can complete the first '`Pc`', and `\hfuzz` is set equal to 1200 pt. The characters `lxxxviii` now begin a paragraph. The `\fi` takes the `\ifdim` out of TEX's condition stack.

(The appendix below gives further information. Examples like this give some glimmering of the weird maneuvers that can be found in the `TRIP` test, an intricate pattern of unlikely code that is used to validate all implementations of TEX.)

19. If, for example, `\thickmuskip` has the value `5mu plus 5mu` that plain TEX gives it, the first command changes its value to `-5mu plus -5mu`, because *scan_glue* in §461 will call *scan_something_internal* with the second argument *true*; this will cause all three components of the glue to be negated (see §431).

The second command, on the other hand, tells TEX to expand '`\the\thickmuskip`' into a sequence of characters, so it is equivalent to

```
\thickmuskip=-5mu plus 5mu
```

(The minus sign doesn't carry into the stretch component of glue, since §461 applies *negate* only to the first dimension found.)

This problem points out a well-known danger that is present in any text-macro-expanding system.

20. We'd have a funny result that two macro texts would be considered to match by `\ifx` unless the first one (the one starting at q when we begin §508) is a proper prefix of the second. (Notice the statement '$p \leftarrow null$' inside the **while** loop.)

21. Because the byte in *dvi_buf*[*dvi_ptr* − 1] is usually not an operation code, and it just might happen to equal *push*.

22. $2_y\,7_d\,1_d\,8_z\,2_y\,8_z\,1_d\,8_z\,2_y\,8_z\,4_y\,5_z\,9_d\,0_d\,4_y\,5_z$.

23. TEX is in 'no mode' only while processing `\write` statements, and the mode is printed during `\write` only when *tracing_commands* > 1 during *expand*. We might think that `\catcode` operations are necessary, so that the left and right braces for `\write` exist; but it's possible to let TEX's error-recovery mechanism supply them! Therefore the shortest program that meets the requirements is probably the following one based on an idea due to Ronaldo Amá, who suggests putting

```
\batchmode\tracingcommands2
  \immediate\write!\nomode
```

into a file. (Seven tokens total.)

24. When *error* calls *get_token*, because the user has asked for tokens to be deleted (see §88), a second level of *error* is possible, but further deletions are disallowed (see §336 and §346). However, insertions are still allowed, and this can lead to a third level of *error* when *overflow* calls *succumb*.

For example, let's assume that *max_in_open* $= 6$. Then you can type '`` \catcode`?=15 \x ``' and respond to the undefined control sequence error by saying '`i\x??`' six times. This leads to a call of *error* in which six '`<insert>`' levels appear; hence *in_open* $= 6$, and one more insertion will be the last straw. At this point, type '`1`'; this enters *error* at a second level, from which '`i`' will enter *error* a third time. (The run-time stack now has *main_control* calling *get_x_token* calling *expand* calling *error* calling *get_token* calling *get_next* calling *error* calling *begin_file_reading* calling *overflow* calling *error*.)

25. In §38, define *str_number* to be the same as *pool_pointer*, and define *str_end* $= 128$. In §39, delete the declaration of *str_start*. In §40, declare

```
function length(s : str_number): integer;
  var t: pool_pointer;
  begin t ← s;
  while str_pool[t] ≠ str_end do incr(t);
  length ← t − s;
  end;
```

In §41, define *cur_length* $\equiv$ (*pool_ptr* $-$ *str_ptr*). In §43, declare

```
function make_string: str_number;   { current string enters the pool }
  var t: str_number;   { the result }
  begin str_room(1); append(str_end);
  t ← str_ptr; str_ptr ← pool_ptr; make_string ← t;
  end;
```

In §44, we can

```
  define flush_string ≡ begin repeat decr(str_ptr);
        until str_pool[str_ptr − 1] = str_end;
        pool_ptr ← str_ptr;
        end
```

The comparison function in §45 is used only in §259, where we can replace

'**if** *length*(*text*(p)) $= l$ **then if** *str_eq_buf*(*text*(p), j)'

by '**if** *str_eq_buf*(*text*(p), j, l)'. The function now has three parameters:

```
function $str\_eq\_buf(s : str\_number; k, l : integer)$: $boolean$;
        { test equality of strings }
  label $exit$;
  var $j$: $pool\_pointer$;  { running index }
  begin $j \leftarrow s$; $s \leftarrow s + l$;
  if $str\_pool[s] \neq str\_end$ then $str\_eq\_buf \leftarrow false$
  else begin while $j < s$ do
       begin if $str\_pool[j] \neq buffer[k]$ then
         begin $str\_eq\_buf \leftarrow false$; $return$; end;
       $incr(j)$; $incr(k)$;
       end;
    $str\_eq\_buf \leftarrow true$;
    end;
exit: end;
```

The procedure of §46 is modified in an obvious, similar way.

The first three statements of §47 become just two: '*pool_ptr* ← 128; *str_ptr* ← 128'. The body of the **for** loop in §48 becomes just

```
if (⟨Character $k$ cannot be printed 49⟩) then
  if $k < '100$ then $str\_pool[k] \leftarrow k + '100$
  else $str\_pool[k] \leftarrow k - '100$
else $str\_pool[k] \leftarrow k$
```

In §59, variable j is no longer needed. If $0 \leq s < 128$ and if s isn't the current new-line character, we now say

```
begin if $str\_pool[s] \neq s$ then
  begin $print\_char("\hat{}")$; $print\_char("\hat{}")$;
  end;
$print\_char(str\_pool[s])$;
end
```

In the other case, where $s \geq 128$, we say

```
while $str\_pool[s] \neq str\_end$ do
  begin $print\_char(str\_pool[s])$; $incr(s)$;
  end
```

In §407, similarly, variable k is eliminated; the loop on k becomes a loop on s, **while** $str\_pool[s] \neq str\_end$.

In §464, replace the two occurrences of '*str_start*[*str_ptr*]' by '*str_ptr*'.

The first loop in §603 becomes

```
$k \leftarrow font\_area[f]$;
while $str\_pool[k] \neq str\_end$ do
  begin $dvi\_out(str\_pool[k])$; $incr(k)$;
  end
```

and the second is like unto it.

26. Let's assume that we have a machine in which *str_pool* is addressed by byte number, so that 8-bit values take no more space than 7-bit values. Method (a) requires us to impose a limit on the length of strings: 255 characters max. This isn't unreasonable, because the only important use of longer strings is in the implementation of `\special`, when the restriction doesn't actually apply (since §1368 doesn't call *make_string*). But method (a) saves no space and little or no time by comparison with the simpler method of problem 25. Problem 25 saves about one byte per string, compared to the text's way. Method (b) saves another byte per string but at the expense of considerable programming complexity; it requires awkward special-casing to deal with empty strings.

27. We'd replace '*width*(*g*)' by

$$width(g) + shift\_amount(g)$$

(twice). Similar changes would be needed in §656. (But a box shouldn't be able to retain its *shift_amount*; this quantity is a property of the list the box is in, not a property of the box itself.)

28. The final line has infinite stretchability, since plain TEX sets `\parfillskip=0pt plus 1fil`. Reports of loose, tight, underfull, or overfull boxes are never made unless $o = normal$ in §658 and §664.

29. If a vbox is repackaged as an hbox, we get really weird results because things that were supposed to stack up vertically are placed together horizontally. The second change would be a lot less visible, except in characters like V where there is a large italic correction; the character would be centered without taking its italic correction into account. (The italic correction in math mode is the difference between horizontal placement of superscripts and subscripts in formulas like V_2^2.)

30. The spacing can be found by saying

`$x==1$ $x++1$ $x,,1$ \tracingall\showlists.`

Most of the decisions are made in §766, using the spacing table of §764. But the situation is trickier in the case of `+`, because a *bin_noad* must be preceded and followed by a noad of a suitable class. In the formula `$x++1$`, the second `+` is changed from *bin_noad* to *ord_noad* in §728. It turns out that thick spaces are inserted after the x and before the 1 in '$x == 1$'; medium spaces are inserted before each + sign in '$x + +1$'; thin spaces are inserted after each comma in '$x,,1$'.

31. The behavior of the simpler algorithm, which we may call Brand X, can be deduced from the demerits values ('`d=`') in the trace output. There is only one reasonable choice, `@@1`, for the first line; and there's only one, `@@2`, for the second. But for the third, a line from `@@2` to `@@3` (the break after '`para-`') has 46725 demerits, which certainly looks worse than the 1225 demerits from `@@2` to `@@4`. This, however, leads Brand X into a trap, since there's no good way to continue from `@@4`. Similarly, Brand X will choose to go from `@@7` to `@@9`, and this forces it to `@@11` and then infelicitously to `@@13` (because the syllable '`break-`' is too long to be squeezed in). The resulting paragraph, as typeset by Brand X, looks like this (awful):

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 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. (This exercise takes awhile, but the data structures are especially interesting; the hyphenation algorithm is a nice little part of the program that can be studied in isolation.) The following tables are constructed:

	op	*char*	*link*
trie[96]	0	96	1
trie[97]	0	97	5
trie[98]	0	97	2
trie[100]	1	98	3
trie[102]	1	99	4
trie[103]	0	98	6
trie[105]	3	99	4

	[1]	[2]	[3]
hyf_distance	2	0	3
hyf_num	1	3	2
hyf_next	0	0	2

Given the word `aabcd`, it is interesting to watch §923 produce the hyphenation numbers '$_0\mathtt{a}_0\mathtt{a}_2\mathtt{b}_1\mathtt{c}_0\mathtt{d}_3$' from this trie.

33. The idea is to keep line numbers on the save stack. Scott Douglass has observed that, although TeX is careful to keep *cur_boundary* up to date, nothing important is ever done with it; hence the *save_index* field in level-boundary words is not needed, and we have an extra halfword to play with! (The present data structure has fossilized elements left over from old incarnations of TeX.) However, line numbers might get larger than a halfword; it seems better to store them as fullword integers.

This problem requires changes to three parts of the program. First, we can extend §1063 as follows:

```
⟨Cases of main_control that build boxes and lists 1056⟩ +≡
non_math(left_brace): begin saved(0) ← line; incr(save_ptr); new_save_level(simple_group);
  end; {the line number is saved for possible use in warning message}
any_mode(begin_group): begin saved(0) ← line; incr(save_ptr); new_save_level(semi_simple_group);
  end;
any_mode(end_group): if cur_group = semi_simple_group then
    begin unsave; decr(save_ptr); {pop unused line number from stack}
    end
  else off_save;
```

A similar change is needed in §1068, where the first case becomes

```
simple_group: begin unsave; decr(save_ptr);   { pop unused line number from stack }
  end;
```

Finally, we replace lines 6–11 of §1335 by code for the desired messages:

```
while cur_level > level_one do
  begin print_nl("("); print_esc("end␣occurred␣when␣");
  case cur_group of
  simple_group: print_char("{");
  semi_simple_group: print_esc("begingroup");
  othercases confusion("endgroup")
  endcases;
  print("␣on␣line␣"); unsave; decr(save_ptr); print_int(saved(0)); print("␣was␣incomplete)");
  end;
while cond_ptr ≠ null do
  begin print_nl("("); print_esc("end␣occurred␣when␣"); print_cmd_chr(if_test, cur_if);
```

34. First, §2 gets a new paragraph explaining what TEXX is, and the banner line changes:

```
define banner ≡ ´This␣is␣TeXX,␣Version␣2.2´   { printed when TeX starts }
```

Then we add two new definitions in §134:

```
define is_xchar_node(#) ≡ (font(#) = font_base)   { is this char_node extended? }
define bypass_xchar(#) ≡
        if is_xchar_node(#) then # ← link(#)
```

(It's necessary to say *font_base* here instead of *null_font*, because *null_font* isn't defined until later.)

The *short_display* routine of §174 can treat an \xchar like an ordinary character, because *print_ASCII* makes no restrictions. Here is one way to handle the change:

```
procedure short_display(p : integer);   { prints highlights of list p }
  label done;
  var n: integer;   { for replacement counts }
    ext: integer;   { amount added to character code by xchar }
  begin ext ← 0;
  while p > mem_min do
    begin if is_char_node(p) then
      begin if p ≤ mem_end then
        begin if is_xchar_node(p) then
          begin ext ← 256 * (qo(character(p))); goto done;
          end;
        if font(p) ≠ font_in_short_display then
          begin if (font(p) < font_base) ∨ (font(p) > font_max) then print_char("*")
          else ⟨ Print the font identifier for font(p) 267 ⟩;
          print_char("␣"); font_in_short_display ← font(p);
          end;
```

```
            print_ASCII(ext + qo(character(p)));   ext ← 0;
            end;
          end
        else ⟨Print a short indication of the contents of node p 175⟩;
      done: p ← link(p);
        end;
      end;
```

A somewhat similar change applies in §176:

```
  procedure print_font_and_char(p : integer);   { prints char_node data }
    label reswitch;
    var ext: integer;   { amount added to character code by xchar, or −1 }
    begin ext ← −1;
  reswitch: if p > mem_end then print_esc("CLOBBERED.")
    else begin if is_xchar_node(p) then
          begin ext ← qo(character(p)); p ← link(p); goto reswitch; end;
      if (font(p) < font_base) ∨ (font(p) > font_max) then print_char("*")
      else ⟨Print the font identifier for font(p) 267⟩;
      print_char("␣");
      if ext < 0 then print_ASCII(qo(character(p)))
      else begin print_esc("xchar"); print_hex(ext * 256 + qo(character(p)));
        end;
      end;
    end;
```

(These routines must be extra-robust.) The first line of code in §183 now becomes

```
    if is_char_node(p) then
      begin print_font_and_char(p);
      bypass_xchar(p);
      end
```

In §208 we introduce a new operation code,

```
    define xchar_num = 17
                { extended character ( \xchar ) }
```

Every opcode that follows it in §208 and §209, from *math_char_num* to *max_command*, must be increased by 1. We also add the following lines to §265 and §266, respectively:

```
        primitive("xchar", xchar_num, 0);
      xchar_num: print_esc("xchar");
```

This puts the new command into TEXX's repertoire.

The next thing we need to worry about is what to do when \xchar occurs in the input. It's convenient to add a companion procedure to *scan_char_num* in §435:

```
  procedure scan_xchar_num;
    begin scan_int;
    if (cur_val < 0) ∨ (cur_val > 65535) then
      begin print_err("Bad␣character␣code");
      help2("An␣\xchar␣number␣must␣be␣between␣0␣and␣255.")
      ("I␣changed␣this␣one␣to␣zero."); int_error(cur_val); cur_val ← 0;
      end;
    end;
```

Similarly, *new_character* gets a companion in §582:

```
function new_xchar(f : internal_font_number; c : integer): pointer;
  var p, q: pointer;  { newly allocated nodes }
  begin q ← new_character(f, c mod 256);
  if q = null then new_xchar ← null
  else begin p ← get_avail; font(p) ← font_base; character(p) ← qi((c div 256)); link(p) ← q;
    new_xchar ← p;
    end;
  end;
```

Extended characters can be output properly if we replace the opening lines of the code in §620 by these:

```
reswitch: if is_char_node(p) then
    begin synch_h; synch_v;
    repeat if is_xchar_node(p) then
        begin f ← font(link(p));
        if character(p) = qi(0) then p ← link(p);  { bypass zero extension }
        end
      else f ← font(p);
      c ← character(p);
      if f ≠ dvi_f then ⟨Change font dvi_f to f 621⟩;
      if is_xchar_node(p) then
        begin dvi_out(set1 + 1); dvi_out(qo(c)); p ← link(p); c ← character(p);
        end
      else if c ≥ qi(128) then dvi_out(set1);
      dvi_out(qo(c));
```

Many of the processing routines include a statement of the form 'f ← *font*(#)', which we want to do only after bypassing the first half of an extended character. This can be done by inserting the following statements:

bypass_xchar(*p*)	in §654;
bypass_xchar(*s*)	in §842;
bypass_xchar(*cur_p*)	in §867;
bypass_xchar(*s*)	in §871;
bypass_xchar(*p*)	in §1147.

In §841 we need to do a little more than a simple bypass:

```
if is_char_node(v) then
  begin if is_xchar_node(v) then
    begin v ← link(v); decr(t);
        { an xchar counts as two chars }
    end;
```

Two changes are needed in order to suppress hyphenation in words that contain extended characters. First we insert

```
if hf = font_base then goto done1;
        { is_xchar_node(s) }
```

after the third line of §896. Then we replace '**endcases**;' in §899 by

```
  endcases
else if is_xchar_node(s) then goto done1;
```

If `\xchar` appears in math mode, we want to recover from the error by including *mmode* + *xchar_num* in the list of cases in §1046. If `\xchar` appears in vertical mode, we want to begin a paragraph by including *vmode* + *xchar_num* in the second list of cases in §1090.

But what if `\xchar` appears in horizontal mode? To handle this, we might as well rewrite §1122:

1122. We need only two more things to complete the horizontal mode routines, namely the `\xchar` and `\accent` primitives.

```
⟨Cases of main_control that build boxes and lists 1056⟩ +≡
hmode + xchar_num: begin scan_xchar_num; link(tail) ← new_xchar(cur_font, cur_val);
  if link(tail) ≠ null then tail ← link(link(tail));
  space_factor ← 1000;
  end;
hmode + accent: make_accent;
```

Finally, we need to extend *make_accent* so that extended characters can be accented. (Problem 34 didn't call for this explicitly, but TEXX should surely do it.) This means adding a new case in §1124:

```
else if cur_cmd = xchar_num then
    begin scan_xchar_num; q ← new_xchar(f, cur_val);
    end
```

and making changes at the beginning and end of §1125:

```
⟨Append the accent with appropriate kerns, then set p ← q 1125⟩ ≡
  begin t ← slant(f)/float_constant(65536);
  if is_xchar_node(q) then i ← char_info(f)(character(link(q)))
  else i ← char_info(f)(character(q));
  w ← char_width(f)(i);
            ⋮
  subtype(tail) ← acc_kern; link(p) ← tail;
  if is_xchar_node(q) then   { in this case we want to bypass the xchar part }
    begin tail_append(q); p ← link(q);
    end
  else p ← q;
  end
```

35. The main reason for preferring the method of problem 34 is that the italic correction operation (§1113) would be extremely difficult with the other scheme. Other advantages are: (a) Division by 256 is needed only once; TEXX's main loops remain fast. (b) Comparatively few changes from TEX itself are needed, hence other ripoffs of TEX can easily incorporate the same ideas. (c) Since fonts don't need to be segregated into 'oriental' and 'occidental', `\xchar` has wide applicability. For example, it gives users a way to suppress ligatures and kerns; it allows large fonts to have efficient 256-character subsets of commonly-used characters. (d) The conventions of TEXX match those of the GF files produced by METAFONT.

The only disadvantage of the TEXX method is that it requires all characters whose codes differ by multiples of 256 to have the same box size. But this is a minor consideration.

Appendix

The solution to problem 18 refers to a special version of TEX called DemoTEX, which allows users to see more details of the scanning process. DemoTEX is formed by making a few changes to parts 24–26 of TEX.

First, in §341, the following code is placed between '*exit:*' and '**end**':

```
if tracing_stats > 2 then
  begin k ← trace_depth; print_nl("");
  while k > 0 do
    begin print("␣"); decr(k);
    end;
  print("|"); print_char("␣");
  if cur_cs > 0 then
    begin print_cs(cur_cs);
    print_char("=");
    end;
  print_cmd_chr(cur_cmd, cur_chr);
  end;
```

(A new global variable, *trace_depth*, is declared somewhere and initialized to zero. It is used to indent the output of DemoTEX so that the depth of subroutine nesting is displayed.)

At the beginning of *expand* (in §366), we put the statements

```
incr(trace_depth);
if tracing_stats > 2 then print("␣<x");
```

this prints '<x' when *expand* begins to expand something. The same statements are inserted at the beginning of *scan_int* (§400), *scan_dimen* (§448), and *scan_glue* (sec461), except that *scan_int* prints '<i', *scan_dimen* prints '<d', and *scan_glue* prints '<g'. (Get it?) We also insert complementary code at the end of each of these procedures:

```
decr(trace_depth);
if tracing_stats > 2 then print_char(">");
```

this makes it clear when each part of the scanner has done its work.

Finally, *scan_keyword* is instrumented in a similar way, but with explicit information about what keyword it is seeking. The code

```
incr(trace_depth);
if tracing_stats > 2 then
  begin print("␣<`"); print(s);
  print_char("´");
  end;
```

is inserted at the beginning of §407, and

```
  if tracing_stats > 2 then print_char("*");
exit: decr(trace_depth);
  if tracing_stats > 2 then print_char(">");
  end;
```

replaces the code at the end. (Here '*' denotes 'success': the keyword was found.)

For example, here's the beginning of what DemoTEX prints out when scanning the right-hand side of the assignment to \hfuzz in problem 18:

```
|! the character = <d
 |! the character 1 <i
  |! the character 1
  |! the character 0
  |! the character 0
  |! the letter P>
 |! the letter P <'em'
  |! the letter P> <'ex'
  |! the letter P> <'true'
  |! the letter P> <'pt'
  |! the letter P
  |! \ifdim =\ifdim <x <d
    |! the character 1 <i
     |! the character 1
     |! the character 2
     |! the letter p>
    |! the letter p <'em'
     |! the letter p> <'ex'
     |! the letter p> <'true'
     |! the letter p> <'pt'
     |! the letter p
     |! the letter t*>
    |! the character =>
```

(After seeing '=', TEX calls *scan_dimen*. The next character seen is '1'; *scan_dimen* puts it back to be read again and calls *scan_int*, which finds '100', etc. This output demonstrates the fact that TEX frequently uses *back_input* to reread a character, when it isn't quite ready to deal with that character.)

Acknowledgement

I wish to thank the brave students of my experimental class for motivating me to think of these questions, for sticking with me when the questions were impossible to understand, and for making many improvements to my original answers.

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

Webless Literate Programming

Jim Fox

Abstract

This article introduces c-~~web~~ (*no-web*, for short) as an alternative to the CWEB 'literate programming' system. c-~~web~~ is a method which allows a programmer to both `tex` (format) and `cc` (compile) the same source, without the need for preprocessors.

What is c-~~web~~

In **c** all comments begin with the characters '/*' and end with the characters '*/'. c-~~web~~ is a macro package that TEXs all comments, 'verbatims' all the code, and uses the comment delimiters to switch between the two modes. A c-~~web~~ program can be compiled directly by **c** and can be formatted directly by TEX. It has the advantage of high portability, while providing fully TEX'd comments, page headers and footers, and a table of contents.

`\title{ ... }` Titles the program.

`\section{ ... }` Begins a section. The section title is also included in the table of contents and in the page header.

`\subsection{ ... }` Begins a subsection. The subsection title is also included in the table of contents.

`\subsubsection{ ... }` Begins a subsubsection.

`\newpage` Causes a page eject after the current line. This is usually used in a comment by itself, e.g., `/* \newpage */`.

`\endc` Ends the c-~~web~~ listing. This is usually the last line in the file, e.g., `/* \endc */`.

`\" ... "` Prints **bold** text.

`\' ... '` Prints *italic* text.

`\| ... |` Prints `typewriter` text.

`*< ... >*` Prints verbatim. This allows **c** code to be included in comments.

Figure 1: c-~~web~~ definitions

Why c-~~web~~?

CWEB is essentially a **c** implementation of Edsger Dijkstra's *Notes on Structured Programming*, with fine formatting thrown in for good measure. The benefits of WEB are well known but it is unsuitable for many programmers and applications for a couple of reasons.

The first problem concerns portability. A program written in CWEB can only be conveniently implemented on a computer which already runs TEX. That is unfortunately a very small subset of the computing world. Anyone writing in CWEB greatly limits the portability of his or her programs.

The second problem concerns the translation of the code part of a program. A well written program consists of small pieces of code consisting of a documentation part, which explains to humans what the part does and how it does it, and a code part, which is a realization of the documentation. Both CWEB and c-~~web~~ print program listings assuming this method, and they both TEX the commentary. Where they differ is in the formatting of the code. c-~~web~~ leaves it alone except for indentation. CWEB gratuitously translates it into something that looks more like mathematics. Because programs undergo continual modifications, one tends to look to the source file to see what the code actually does. Many programmers, myself included, are more comfortable with code in the file that looks like the code in the listing.

```
/* sample.c in cnoweb format
   by Jim Fox, August 19, 1990
   \input cnoweb
   \title {Sample with procedure} */
        .
        .
/* \section{Sum}
    This procedure computes and returns
    $$ {\bf sum} = \sum_{i=0}^{\bf n}
       {\bf f}(i)$$

    There is no error checking
    in this example. */

double sum(f,n)
double (*f)();    /* function to call */
int n;            /* summation limit */
{
  int i;
  double s = 0;

   for (i=0; i<=n; i++) {
    s += f(i);
   }
  return (s);
}
    .
    .
/* \endc */
```

Figure 2: Procedure **sum** from program **sample**.

CWEB allows a programmer to break programs into small pieces without resorting to **c**'s procedure calls. This is an attempt to directly implement the 'layers' described by Dijkstra. c-~~web~~ cannot do this. However, procedures are often the better choice. They are more easily tested, more formally isolated from the caller, and usually produce more flexible code.

In any case, my effort here is only to introduce an alternate 'literate programming' method—not to compare the two beyond this introduction.

Using c-~~web~~

The c-~~web~~ program must begin with a comment that contains:

```
\input cnoweb
```

and must end with a comment that contains:

```
\endc
```

sum(f,n) **sample – 9**

/* **sum(f,n)** This procedure computes and returns

$$\mathbf{sum} = \sum_{i=0}^{\mathbf{n}} \mathbf{f}(i)$$

There is no error checking in this example. */

```
double sum(f,n)
double (*f)();       /* function to call */
int n;               /* summation limit */
{
    int i;
    double s = 0;


    for (i=0; i<=n; i++) {
        s += f(i);
    }
    return (s);
}
```

Figure 3: **sum** from the listing of **sample.c**

Other than this the program need not contain any TeX text. Most programs, however, will use plain TeX commands in comments, as well as several new commands provided by c-~~web~~. These are described in Figure 1.

Figure 2 is a sample procedure, **sum**, from a program in c-~~web~~ format. Figure 3 is the listing of the procedure. Not shown in figure 3 is the title page, which includes the title, synopsis (none in this example), and table of contents.

Features of c-~~web~~, some of which are demonstrated in the example, include:

1. Page breaks occur only before comments.
2. The code portion is printed not quite verbatim—indentation is automatically provided. Lines following an opening bracket or parenthesis are indented until the line containing the closing bracket or indentation.
3. The page heading contains the **c** file name, the page number, and the current section name.
4. Alignment rules on the top and left help verify indentation.

Trying it out

A sample program (`pf.c`) demonstrates c-~~web~~ and describes the commands in more detail. Interested persons should obtain a copy of the macro file (`cnoweb.tex`) and the sample program by anonymous ftp to `u.washington.edu`. They are in the directory `pub/tex/cnoweb`. Anyone without access to ftp may request the files by mail to me at the address below.

◇ Jim Fox
University of Washington
`fox@cac.washington.edu`

A TeX Previewer for "Slow" Terminals

Harold T. Stokes

In our department, we have a multi-user computer cluster with one copy of TeX and one laser printer. Our previewer sends document pages to the user's terminal in the Tektronix 4010/4014 graphics format. Most graphics terminals can emulate the Tektronix 4010/4014. This includes PCs using a terminal emulator like MSKermit.

The terminals in our department are connected to the system through a data switch. The rate at which data can be sent to a terminal is limited to 9600 baud (1200 bytes/second). This is rather slow for displaying graphics.

As an example, consider how we might draw the character T on the screen. The individual pixels for this character (from the `cmr10` font) are shown in Fig. 1*a*. Since Tektronix 4010/4014 graphics is vector-oriented, we might try the obvious raster scan shown in Fig. 1*b*. However, there are 44 line segments in that raster scan. To draw a single line segment in Tektronix 4010/4014 graphics, at least seven bytes must be sent to the terminal (sometimes eight or nine). At 9600 baud, it would require at least 0.25 seconds to display this single character. An entire document page which may contain 3000 characters would require more than ten minutes to be displayed. This, of course, is unacceptable for a previewer.

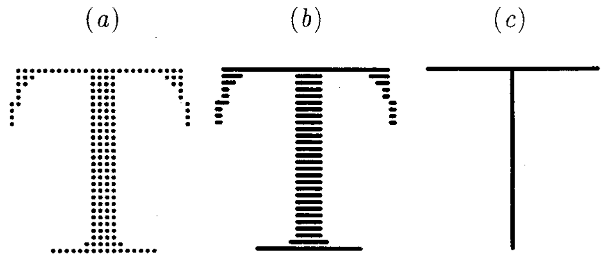
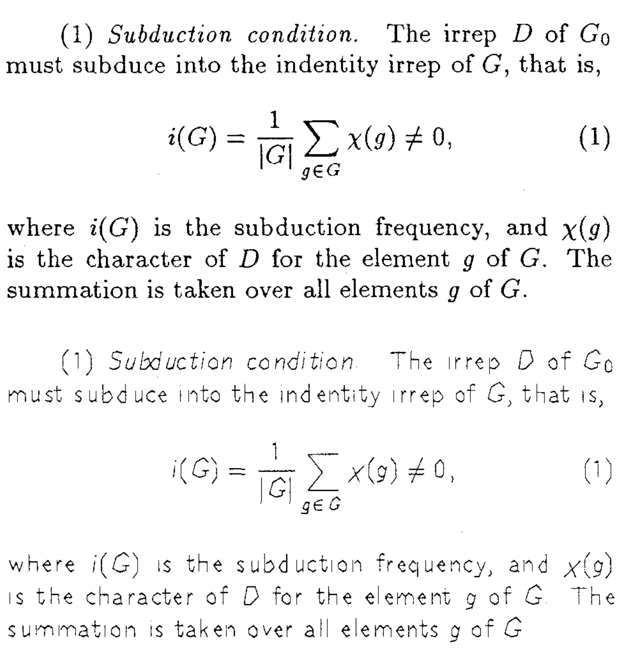


Fig. 1. The character T from `cmr10`: (*a*) pixels, (*b*) raster scan, (*c*) stick figure.

We solved this problem by designing a new set of fonts. We call them "stick-figure" fonts. The T is displayed as two line segments (Fig. 1*c*). In Fig. 2 we show a sample of some text, including a displayed equation. (The actual clarity will depend on the resolution of the graphics screen used.) Using the stick-figure font, a page containing 3000 characters can usually be displayed on the screen in less than a minute. Some users have even used this previewer from home over a modem.



(1) *Subduction condition.* The irrep D of G_0 must subduce into the indentity irrep of G, that is,

$$i(G) = \frac{1}{|G|} \sum_{g \in G} \chi(g) \neq 0, \qquad (1)$$

where $i(G)$ is the subduction frequency, and $\chi(g)$ is the character of D for the element g of G. The summation is taken over all elements g of G.

(1) *Subduction condition* The irrep D of G_0 must subduce into the indentity irrep of G, that is,

$$i(G) = \frac{1}{|G|} \sum_{g \in G} \chi(g) \neq 0, \qquad (1)$$

where $i(G)$ is the subduction frequency, and $\chi(g)$ is the character of D for the element g of G The summation is taken over all elements g of G

Fig. 2. Sample TEX output (*a*) from our laser printer and (*b*) how it appears using our previewer with stick-figure fonts.

Our previewer has one major fault. We cannot distinguish between some of the different fonts. But the speed gained more than compensates for this disadvantage.

Our previewer is written in a portable C computer language.

◇ Harold T. Stokes
Department of Physics and Astronomy
Brigham Young University
Provo, Utah 84602
Phone: (801) 378-2215
Bitnet: `stokesh@byuvax`

Philology

Report on Multilingual Activities

Michael J. Ferguson

We now have an Extended TEX Font Encoding which should be officially approved by TUG and available on the archives by the time you read this report. This new standard is the first step in exploiting the enhanced capabilities of TEX 3.0 in a multilingual environment. The use of this standard will enable the direct use of accented characters, hyphenation, and will create, for the first time, a consistent interface to all text fonts in TEX systems. The new standard is intended to be used for all of TEX's text fonts. This means that accessing a particular code in any font will produce similar results. To this end, a **tt** font will have a **ffi** sequence of monospaced characters at the same location as the **ffi** ligature in the bold roman font. It should also serve as the encoding standard for non-Greek variable names in math mode.

The new standard extends the alphabetic content of ISO Latin 1 by including all of the linguistic characters in ISO Latin 2 along with ligatures and punctuation relevant to these languages. This standard includes the national characters, without the necessity of explicit accenting, for Albanian, Czech, Danish, Dutch, English, Faeroese, Finnish, French, German, Hungarian, Icelandic, Irish, Italian, Norwegian, Polish, Portugese, Rumanian, Serbocroatian, Slovak, Slovene, Spanish, Swedish and Turkish. It also includes a block of diacritics, so that these "basic" characters may be extended by use of TEX's accenting mechanism. This capability should allow for the extension to other languages, for example Lappish, at the possible penalty of not allowing hyphenation of words containing these explicitly accented characters.

The standard also includes some important innovative extensions to increase flexibility of use and enhance lexical semantic consistency.

- It includes both a "dash" and an explicit "hyphen char". This capability allows font designers the option of replacing the "-" with an "=" without losing the dash. Since the "-" is no longer the hyphen char, it allows words with explicit dashes, such as INRS-Télécommunications to be hyphenated. It also allows for the Serbocroatian hyphenation rule which says that a word broken at a "-" should have the "-" both at the end of the line and the beginning of the next.

- It includes codes for opening quotes ", closing quotes ", and the ASCII double quote character ". TEX normally accesses the opening and closing quotes as ligatures, and tends to use the ASCII double quote for special purposes such as designating hex numbers. The ASCII double quote code in a font should correspond to straight quotes, much as they are in the `tt` fonts. This makes a new character available in the Roman fonts and removes an irritant in the use of double quotes in these fonts. The font designer can decide whether the opening double quotes, closing double quotes, and the ASCII double quotes are distinguishable. The standard also includes < > in the normal ASCII location.
- It includes a small "o" to match the "%" to enable allow for a "milli" or "milli.milli...", a "visible space", and both diacritic and non-diacritic versions of such characters as the " ^ ~ ".
- Perhaps the most innovative feature of the new standard, and potentially the most powerful, is the inclusion of a "`Compound Word Mark -- <cwm>`". The `<cwm>`, whose image is invisible, is effectively a character of zero width and depth. This character would be used to break up ligatures on subword boundaries in German. For example, the German word "auflage" should not have the "fl" ligature because it occurs on the subword boundary between "auf" and "lage". However a word such as "flach" should have the ligature. There are many ways to enable input of the `<cwm>`. One might be to define the "`\fl` " as meaning the `f<cwm>l`. Then the input sequence "`au\fl age`" would be rendered as "auflage". The `<cwm>` can also be used to selectively enable ligatures. For instance, if one wanted the "ff" ligature in the font to be off most of the time, the ligature access sequence in the font would be "`f<cwm>f`". Explicitly including the `<cwm>` at input would enable the ligature. The `<cwm>` can also serve as an invisible hyphen char. It is our belief that this is just the beginning of the possible uses for this character.

Although we have an excellent extended text font definition, there is much work to be done. Perhaps the most urgent is the definition of an extended math symbol font. A key decision in this particular exercise is whether to include Greek symbols in an extended math symbol font or to access Greek symbols used in math from a Greek text font or from an extended math font. In the interim the Greek characters may be accessed from the current cm fonts.

Now that we have an extended font encoding standard, we are in a position to create complete TEX 3.0 compatible hyphenation patterns. These patterns should be expressed using only the normal printable ASCII character set so that they can be transmitted by electronic mail. All non-printable ASCII characters such as an ç or an Icelandic "thorn" should be expressed as backslash sequences. Thus the French pattern which allows a hyphen to occur before any ç would be encoded as `1\c c` and `\accenthyphcodes` preceding the inputting of these patterns would include `\c c{"e7}` for the ç definition necessary within the patterns. Finally, the new patterns need to account for use of the `<cwm>`. It may be as simple as including the pattern equivalent to `1<cwm>`, or it may be more elaborate. I would like to review these patterns for syntactic consistency but would like the originators to make the arrangements for the actual distribution. I could act as the coordinator of the list of "responsables".

Finally, this new font encoding standard is important for all TEX ports because it defines TEX's internal character codes. Thus this encoding will be the basis of the `xchr[..]` and `xord[..]` translation arrays.

I should especially like to thank Jan Michael Rynning for his work in collecting the detailed information needed for creating this standard. I should also like to thank Jan Michael and Norbert Schwarz for their leadership, hard work, and attention to detail during the Cork meeting that made this important standard possible.

◇ Michael J. Ferguson
Coordinator for Multilingual Activities
INRS - Télécommunications
Université du Québec
3 Place du Commerce
Verdun H3E 1H6
Québec, Canada
`mike@inrs-telecom.uquebec.ca`

	´0	*´1*	*´2*	*´3*	*´4*	*´5*	*´6*	*´7*	
´00x	`	´	ˆ	˜	¨	˝	˚	ˇ	″0x
´01x	˘	¯	˙	¸	˛	‚	<	>	
´02x	“	”	„	≪	≫	–	—	<cwm>	″1x
´03x	₀	ı	ȷ	ff	fi	fl	ffi	ffl	
´04x	␣	!	"	#	$	%	&	’	″2x
´05x	(	)	*	+	,	-	.	/	
´06x	0	1	2	3	4	5	6	7	″3x
´07x	8	9	:	;	<	=	>	?	
´10x	@	A	B	C	D	E	F	G	″4x
´11x	H	I	J	K	L	M	N	O	
´12x	P	Q	R	S	T	U	V	W	″5x
´13x	X	Y	Z	[	\	]	^	_	
´14x	‘	a	b	c	d	e	f	g	″6x
´15x	h	i	j	k	l	m	n	o	
´16x	p	q	r	s	t	u	v	w	″7x
´17x	x	y	z	{	|	}	˜	(hyph.char) -	
´20x	Ă	Ą	Ć	Č	Ď	Ě	Ę	Ğ	″8x
´21x	Ĺ	Ľ	Ł	Ń	Ň	Eng/Ŋ	Ő	Ŕ	
´22x	Ř	Ś	Š	Ş	Ť	Ţ	Ű	Ů	″9x
´23x	Ÿ	Ź	Ž	Ż	IJ	İ	đ	§	
´24x	ă	ą	ć	č	ď	ě	ę	ğ	″Ax
´25x	ĺ	ľ	ł	ń	ň	ŋ	ő	ŕ	
´26x	ř	ś	š	ş	ť	ţ	ű	ů	″Bx
´27x	ÿ	ź	ž	ż	ij	¡	¿	£	
´30x	À	Á	Â	Ã	Ä	Å	Æ	Ç	″Cx
´31x	È	É	Ê	Ë	Ì	Í	Î	Ï	
´32x	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	Œ	″Dx
´33x	Ø	Ù	Ú	Û	Ü	Ý	Þ	SS	
´34x	à	á	â	ã	ä	å	æ	ç	″Ex
´35x	è	é	ê	ë	ì	í	î	ï	
´36x	ð	ñ	ò	ó	ô	õ	ö	œ	″Fx
´37x	ø	ù	ú	û	ü	ý	þ	ß	
	″8	″9	″A	″B	″C	″D	″E	″F	

A few words of explanation:

´000–´014 are accents. *´014* is an ogonek. *´015–´024* are quotation marks. *´030* is a small 0 to put after the per cent sign, to turn it into a per thousand (%₀) or per million (%₀₀) sign. *´027* (cwm) is a compound word mark (a zero-width invisible character) used e.g. for avoiding ligatures. *´040* is a visible space. *´042* is a *straight* double quotation mark.

´041–´176 is like the 7 bit ASCII code. Some characters that — at first glance — appear duplicated as accent characters usually have a different shape.

´177 is the hyphen character (that may be different from the dash (*´055*)). *´202*, *´210*, *´242*, and *´250* are A's and E's with ogonek accents.

The table has been sorted to reflect `\uppercase` `\lowercase` mechanism for all characters.

This table shows the character codes' positions, but the shapes are only approximations.

Fonts

Filenames for Fonts

Karl Berry

As more typeface families become available for use with TeX, the need for a consistent, rational naming scheme for the font filenames concomitantly grows. Some (electronic) discussion has gone into the following proposal; I felt it was appropriate now to bring it before a wider community. In some respects, it follows and simplifies Mittelbach's and Schöpf's article in *TUGboat*, volume 11, number 2 (June 1990).

Here are some facts about fonts that went into the hopper when creating this proposal:

- TeX runs on virtually all computers, under almost as many operating systems, all with their own idea of how files should be named. Any proposal regarding filenames, therefore, must cater to the lowest common denominator. That seems to be eight characters in length, not counting any extension, and with case being insignificant. Characters other than letters and numerals are probably unusable.
- Most typefaces are offered by several vendors. The version offered by vendor A is not compatible with that of vendor B.
- Typefaces typically come in different weights (hairline to extra heavy), different expansions (ultra condensed to wide), and an open-ended range of variants (italic, sans serif, typewriter, shadow, ...). No accepted standards exist for any of these qualities, nor are any standards ever likely to gain acceptance.[1]
- The Computer Modern typeface family preserves traditional typesetting practice in at least one important respect: different sizes of the same font are not scaled linearly. This is in contrast to most commercial fonts available.

[1] Editor's note: A draft international standard, ISO/IEC DIS 9541, Font Information Interchange, attempts to define these qualities, among others, in a manner acceptable to font suppliers and usable by a wide variety of typesetting software, including (indirectly) TeX. The Editor has been a participant in this working group for several years, and is trying to make sure that the needs of TeX users are heard.

Here is how I propose to divide up the eight characters:

```
FTTWVEDD
```

where

- `F` represents the foundry that produced the font, and is omitted if there isn't one.
- `TT` represents the typeface name.
- `W` represents the weight.
- `V` represents the variant, and is omitted if both it and the expansion are "normal".
- `E` represents the expansion, and is omitted if it is "normal".
- `DD` represents the design size, and is omitted if the font is linearly scaled from a single `tfm` file.

See the section on virtual fonts (towards the end) for an exception to the above.

The weight, variant, and expansion are probably all best taken from the original source of the typeface, instead of trying to relate them to some external standard.

Before giving the lists of abbreviations, let me point out two problems, to neither of which I have a good solution. 1) Assuming that only the English letters are used, two letters is enough for only 676 typeface families (even assuming we want to use all possible combinations, which is doubtful). There are many more than 676 typeface families in the world. 2) Fonts with design sizes over 100 pt are not common, but neither are they unheard of.

On to the specifics of the lists. If you adopt this proposal at your own installation, and find that you have fonts with some property I missed, please write to me (see the end of the article for various addresses), so I can update the lists. You can get the most up-to-date version of these lists electronically, by anonymous ftp from the host `ftp.cs.umb.edu`. I will also send them to you by electronic mail, if necessary.

I give the letters in lowercase, which is preferred on systems where case is significant. The lists are in alphabetical order by the abbreviations.

Foundry

a	Autologic
b	Bitstream
c	Compugraphic
g	Free Software Foundation (g for GNU)
h	Bigelow & Holmes (with apologies to Chuck)
i	International Typeface Corporation
p	Adobe (p for PostScript)

`r`	reserved for use with virtual fonts; see below
`s`	Sun

Typeface families

`ad`	Adobe Garamond
`ag`	Avant Garde
`ao`	Antique Olive
`at`	American Typewriter
`bb`	Bembo
`bd`	Bodoni
`bg`	Benguiat
`bk`	Bookman
`bl`	Balloon
`bv`	Baskerville
`bw`	Broadway
`cb`	Cooper Black
`cl`	Cloister
`cr`	Courier
`cn`	Century
`cs`	Century Schoolbook
`hv`	Helvetica
`gm`	Garamond
`go`	Goudy Oldstyle
`gs`	Gill Sans
`jo`	Joanna
`lc`	Lucida
`lt`	Lutetia
`nc`	New Century Schoolbook
`op`	Optima
`pl`	Palatino
`pp`	Perpetua
`rw`	Rockwell
`st`	Stone
`sy`	Symbol
`tm`	Times
`un`	Univers
`uy`	University
`zc`	Zapf Chancery
`zd`	Zapf Dingbats

Weight

`a`	hairline
`b`	bold
`c`	black
`d`	demi
`h`	heavy
`i`	extra light
`k`	book
`l`	light
`m`	medium
`r`	regular
`s`	semi
`t`	thin
`u`	ultra
`x`	extra bold

In order of lightest to heaviest (more or less):

hairline	demibold
thin	semi
extra light	bold
light	extra bold
book	heavy
regular	black
medium	ultra

Variant

`b`	bright
`c`	small caps
`e`	engraved
`g`	grooved (as in the IBM logo)
`h`	shadow
`i`	(text) italic
`l`	outline
`n`	informal
`o`	oblique (i.e., slanted)
`r`	normal (roman or sans)
`s`	sans serif
`t`	typewriter
`u`	unslanted italic

If the variant is `r`, and the expansion is also normal, both the variant and the expansion are omitted. When the normal version of the typeface is sans serif (e.g., Helvetica), `r` should be used, not `s`. Use `s` only when the typeface family has both serif and sans serif variants.

Expansion

`c`	condensed (by hand)
`e`	expanded (automatic)
`n`	narrow (automatic)
`o`	extra condensed
	regular, normal, medium (always omitted)
`w`	wide
`x`	extended (by hand)

In order of narrowest to widest (more or less):

extra condensed	extended
condensed	expanded
narrow	wide
regular	

Expansion of fonts is sometimes done automatically (as in PostScript **scale**), and sometimes done by humans. I chose 'narrow' and 'expanded' to imply the former, and 'condensed' and 'extended' to imply the latter, as I believe this reflects common usage.

Virtual fonts

In concert with releasing TEX 3.0 and METAFONT 2.7, Don Knuth wrote two new utility programs: VFtoVP and VPtoVF, which convert to and from "virtual" fonts. Virtual fonts provide a general interface between the writers of TEX macros and font suppliers. In general, therefore, it is impossible to come up with a general scheme for naming virtual fonts, since each virtual font is an individual creation, possibly bringing together many unrelated fonts.

Nevertheless, one common case is to use virtual fonts to map TEX's default accent and other character code conventions onto a vendor-supplied font. For example, dvips (by Tom Rokicki) does this for fonts given in the PostScript "standard encoding". In this case, each font consists of a "virtual" tfm file, which is what TeX uses, a "raw" tfm file, which corresponds to the actual device font, and a vf file, which describes the relationship between the two.

This adds another dimension to the space of font names, namely, "virtualness" (or rather, "rawness", since it is the virtual tfm files that the users want to see). But we have already used up all eight characters in the font names.

The best solution I have been able to think of is this: prepend r to the raw tfm files; the virtual tfm files should be named with the usual foundry prefix. For example, the virtual Times Roman tfm file is named ptmr, as usual; the raw Times Roman tfm file is named rptmr. To prevent intolerable confusion, I promise never to give a foundry the letter r.

This scheme will work only as long as the virtualized fonts do not have design sizes; if they do, another foundry letter will have to be allocated, it seems to me.

A pox upon the houses of those who decided on fixed-length filenames!

Examples

In closing, I will give two examples. First, the fonts in the Univers typeface family were assigned numbers by its designer, Adrien Frutiger. (You can see the scheme on, for example, page 29 of *The Art of Typo.icon.ography*, by Martin Solomon.) Naturally, we want to give them names.

45 (light): unl
46 (light italic): unli
47 (light condensed): unlrc
48 (light condensed italic): unlic
49 (light extra condensed): unlro
53 (medium extended): unmrx
55 (medium): unm
56 (medium italic): unmi
57 (medium condensed): unmrc
58 (medium condensed italic): unmic
59 (medium extra condensed): unmro
63 (demibold extended): undrx
65 (demibold): und
66 (demibold italic): undi
67 (demibold condensed): undrc
68 (demibold condensed italic): undic
73 (bold extended): unbrx
75 (bold): unb
76 (bold italic): unbi
83 (extra bold extended): unxrx

Second, here are names for the 35 standard PostScript fonts:

AvantGarde-Book: pagk
AvantGarde-BookOblique: pagko
AvantGarde-Demi: pagd
AvantGarde-DemiOblique: pagdo
Bookman-Demi: pbkd
Bookman-DemiItalic: pbkdi
Bookman-Light: pbkl
Bookman-LightItalic: pbkli
Courier-Bold: pcrb
Courier-BoldOblique: pcrbo
Courier: pcrr
Courier-Oblique: pcrro
Helvetica-Bold: phvb
Helvetica-BoldOblique: phvbo
Helvetica-NarrowBold: phvbrn
Helvetica-NarrowBoldOblique: phvbon
Helvetica: phvr
Helvetica-Oblique: phvro
Helvetica-Narrow: phvrrn
Helvetica-NarrowOblique: phvron
NewCenturySchlbk-Bold: pncb
NewCenturySchlbk-BoldItalic: pncbi

NewCenturySchlbk-Italic: `pncri`
NewCenturySchlbk-Roman: `pncr`
Palatino-Bold: `pplb`
Palatino-BoldItalic: `pplbi`
Palatino-Italic: `pplri`
Palatino-Roman: `pplr`
Symbol: `psyr`
Times-Bold: `ptmb`
Times-BoldItalic: `ptmbi`
Times-Italic: `ptmri`
Times-Roman: `ptmr`
ZapfChancery-MediumItalic: `pzcmi`
ZapfDingbats: `pzdr`

Please contact me if you have any comments or additions.

◇ Karl Berry
135 Center Hill Rd.
Plymouth, MA 02360
`karl@cs.umb.edu`

Arabic, Persian and Ottoman TEX for Mac and PC

Yannis Haralambous

¡El peor con TEX,
es la gente
que lo gusta!
— Anonymous

The whole thing started when my friend Siavash Mirshams Shahshahani from the Sharif University of Technology, Tehran, asked me to find some arabic-alphabet TEX for the PCs of his department. I looked around and found out that there is none, at least not in the public domain. Unfortunately, not even TEX–XET, the right–to–left version of TEX is available for Mac or PC. So I decided to fill the gap. For this I had to create the necessary font `yarb` — which is suitable for arabic, persian and ottoman as well — and a Pascal preprocessor `yarbtex`. This work has been done on a Mac SE/30 with OzTEX, MacMETAFONT, and Think Pascal v3.0.

The arabic alphabet

The common arabic alphabet has the following 28 letters:

ذ د خ ح ج ث ت ب ا
غ ع ظ ط ض ص ش س ز ر
ی و ه ن م ل ك ق ف

To cover also persian and pre-1932 turkish, one has to consider the following 5

ڭ گ ژ چ پ

Finally ڤ is used for the arabic transliteration of the sound "v", as in "Vienna": ڤينا.

Let's see now how letters are combined to form words; with the letters ت, ك and ع we can form the word تكع (right to left!). As you can see, letters are written in a different way according to their position inside the word. Consequently, arabic letters appear in four forms: *initial* (like ﺗ for ت), *middle* (like ﻜ for ك), *final* (like ﻊ for ع) and *isolated*. The letters ا, د, ذ, ر, ز, ژ, و have only final and isolated forms. So when they occur inside a word, the next letter is initial (as in ولد).

As in german, there are short and long vowels. The long ones are ا, و and ى. The short ones are not represented at all, except in classical or educative texts, where they are represented by accents: َ (fatha), ِ (kasra), ُ (damma), respectively for the sounds of a, i and u. In this case, the lack of vowel is represented by ْ (tašdid). Also there are other accents (ٓ, ً, ٌ, ٍ) for special purposes.

The preprocessor `yarbtex`

While typing the input, it would be too tedious to think in which form each letter should be written, and since this problem is much too complicated to be solved only by ligatures inherent to the font, it was necessary to make a preprocessor which converts an `xxx.arb` input file, into a `xxx.tex` TEX-file. The usual procedure

$$\texttt{.tex} \xrightarrow{\TeX} \texttt{.dvi} \xrightarrow{\texttt{dvitops}} \texttt{.ps}$$

is now one step longer

$$\texttt{.arb} \xrightarrow{\texttt{yarbtex}} \texttt{.tex} \xrightarrow{\TeX} \texttt{.dvi} \xrightarrow{\texttt{dvitops}} \texttt{.ps}.$$

Let's take a look now at an `xxx.arb` file: to typeset in arabic, persian and/or ottoman (in short APO) you have to enter into "arabic mode" by typing a vertical bar `|`. The `|` will change to arabic mode regardless of whether you are in text or math mode, so you should use `$\vert$` to obtain | in non-arabic mode.

Once you are in arabic mode, you can choose between the following submodes:

a. *text and arabic numerals* (the numerals used in arabic typography),

b. *transliterated text* (to get an output in latin alphabet, following your own transliteration),
c. *comment mode* (to insert comments),
d. *math mode* (to insert a *short* math sequence) and *display math mode*,
e. *command mode* (to insert *some* commands which have to be executed inside arabic mode).

Text and arabic numerals. APO text has to be written in a special "machine-like" transliteration. By this I mean that you'll have to type as few as possible ASCII characters and that this text won't be very readable (at least until you get used to it). Since there are no uppercase letters in APO I used lower and uppercase ASCII characters to represent different APO letters: `s` stands for س and `S` for ش. I also used "hat + character" combinations: `^s` gives ص and `^S`, چ. Here is an example: to obtain

أَلأَرنب و الفيل

you have to input

```
|^A^Lrnb U AlfIl|
```

which will look like

```
\arbword{\yarb{\char133}{\char237}%
{\char166}{\char12}%
{\char0}}
\arbword{\yarb{\char246}}
\arbword{\yarb{\char231}{\char250}%
{\char206}{\char229}{\char128}}
\arboff
```

in the corresponding `.tex` file. You can find the complete list of input codes in Appendix A (column 3). I suggest you make a copy of this list and keep it in front of you while you are typing; this because, unfortunately, not all characters transliterate into the right phonetic counterparts (like `c` which stands for ة).

APO words are never hyphenated. The way typographical spacing problems are resolved is by inserting straight line segments ـ between characters. Perhaps in some later version of `yarbtex` this will be done automatically. For the moment you'll have to do it manually: just type an `_` to get a line segment with the length of half an n-dash. Here is some advice: if you are using accents, sometimes it looks better to place the accent over the middle of the line segments, حـــَــة, instead of one of the ends, حَـــــة. Note that all the "TeX-forbidden" ASCII characters we encountered (`^`, `_`, etc.) and will encounter, disappear in the `xxx.tex` file.

When you type `0`, `1`,..., `9` inside arabic mode you get the arabic numerals

٠، ١، ٢، ٣، ٤، ٥، ٦، ٧، ٨، ٩.

`yarbtex` automatically writes numbers left-to-right: 143/1962 becomes ١٩٦٢\١٤٣. The punctuation marks . ، : ؛ ؟ ! \ - — ( ) [ ] and special characters ؍ ﷺ ۞ $ are included in the font.

When you exit arabic mode (by typing again a vertical bar `|`), a new paragraph gets started. If you want just to insert a *short* arabic sequence inside latin text, you'll have to type `\ins` before getting into arabic mode. We will return to this later in the discussion of the output file.

Transliterated text. As in the $\mathcal{AMS}$-TeX cyrillic package, you can use the same input to obtain either a text in the arabic alphabet or a transliteration into the roman alphabet. The mechanism is simple: just type `@␣` inside arabic mode, and everything up to the next `@␣`, will be transliterated into latin alphabet.

The natural question to ask is: which transliteration should be chosen? Transliteration is most of the time based on phonetics and these can be very different from one country to the other. So it should be possible to use different transliterations (for example you could use characters from the Washington Computer Modern IPA font `wsuipa`), and switch easily from one to the other, without having to recompile `yarbtex`'s source. The solution I propose is the following: `yarbtex` is accompanied by a text-file, named `yarbtex.dat`. The first $57 \times 4 = 228$ lines of this file contain the transliterations of the 57 APO letters and accents, each in its 4 forms. The transliteration included in the package (and given also in Appendix A, column 2) comes from A. A. Ambros's *Einführung in die moderne arabische Schriftsprache*. Here is how to change an item. Let's say you want to transliterate ـث (the final form of ث) by "th" instead of "ṭ". As you can see in column 4 of Appendix A, ث has the number *5*. Go to line $23 = \mathit{5} \times 4 + \mathbf{3}$ (**0** for isolated, **1** for initial, **2** for middle and **3** for final) of `yarbtex.dat` and replace `\d t` by `th`.

Comment mode. Just use `%` to insert a comment: everything on the right of `%` in the `.arb` file will stay on the right of it in the `.tex` file, in order to be considered as a comment by TeX.

Math mode. If you want to insert a *short* math sequence (such as `$f(x)$`), go ahead and type it. But you should try to avoid having math expressions split on an end-of-line, since it will be rather confusing to split them from left to right when the text goes from right to left. For longer math sequences use display math mode, as in TeX.

Command mode. Let us first describe the output file. Each APO character is written as `{\char` *xxx*`}`. The braces are there to allow the use of accents: `\fatha{\char160}` gives . Each arabic word gets into an `\arbword{...}` macro. Everytime TEX reads this macro, it first checks if its argument fits into the current line. If this is the case, it appends it on the left of the line and looks for the next `\arbword{...}`; else, it outputs the line and starts a new one. When you leave arabic mode, an `\arboff` macro appears in the output file. If the arabic text is to be inserted into a latin text (that is if you had typed `\ins` before entering into arabic mode), `\arboff` just outputs the remaining `\arbwords`; else, it also starts a new paragraph.

Since all these macros deal with `\hboxes`, one should be very careful which macros to insert into arabic mode and which not. For example, a change of arabic font must occur inside each `\arbword` and on the left of the first `{\char }`, but a macro like `\centerline` has to occur outside of arabic mode. The solution is the following: if you need a macro to affect the argument of `\arbword`, type an asterisk `*` and the macro in braces; for example if `\yarbbf` is the boldface arabic font, type `*{\yarbbf}`. This feature is similar to TEX's `\everyhbox`; all forthcoming arabic words will be in boldface, until you type `*`, which is equivalent to `*{}`. If you want to add a second macro `\yyy` you have to rewrite the first: `*{\yarbbf\yyy}`, since each `*{...}` replaces the previous one.

On the other hand, for macros affecting the whole `\arbword` box, you'd better exit arabic mode. For example, the first line in the text of Appendix B (The Rabbit and the Elephant) has been typeset in the following way:

```
\noindent\centerline
{\ins|*{\yarbbig}^A^Lrnb U AlfIl|}
```

The yarb font

The characters of this font belong to the nas̲h̲i style, which is the most common in arabic typography. At this time there is not yet enough metaness in this font; in some later version, the same `.mf` sources will also produce typewriter and straight modern styles.

Calligraphic arabic has many ligatures and is much more difficult to typeset. The ligature problem could be solved, by using more than one table of 256 characters for each font and creating a special version of `yarbtex` which would, beside its regular tasks, detect the ligatures. For the moment, the only ligature I include is the standard lām-ālif لا, which is input by one ASCII character, namely `L` (instead of `lA`).

The ornamentation of Appendix B comes from OSAMA EL NAHAS's beautiful book on *Islamic Decorative Elements* (Cairo 1985). The repetitive and symmetric character of islamic decoration makes it a real challenge for TEX and METAFONT.

The standard `yarb` font has 256 characters, but since there are still PCs which don't support the larger fonts, there is a second version of `yarbtex` which uses two different 128-character fonts `yarb` and `ysarb`.

Things remaining to do and availability

Besides further development on calligraphy and decoration, all other TEX features will be made available in APO. Compatibility with macro-packages like AMS-TEX, LAMS-TEX, LATEX will be tested, and fine points of APO typography studied. I would be really grateful for every suggestion and comment.

The status of the `YARB` package is postcard-ware (which means that each user should send me a postcard, for my collection). It is freely available at my address either by electronic mail, or by ordinary mail (please add some answering coupons).

◇ Yannis Haralambous
Université de Lille 1
59655 Villeneuve d'Ascq
France
Bitnet: `yannis@frcitl81`

Output	Transl. Output	Input	N	Output	Transl. Output	Input	N
ا	ā	A	1	م	m	m	30
ب	b	b	2	ن	n	n	31
پ	p	p	3	ه	h	h	32
ت	t	t	4	ة	(tāʒ marbūṭa)	c	33
ث	ṯ	T	5	و	ū	U	34
ج	ğ	Z	6	ى	ī	I	35
چ	tsch	^S	7	لا	lā	L	36
ح	ḥ	H	8	ـَ	(fatḥa)	a	37
خ	ẖ	j	9	ـِ	(kasra)	i	38
د	d	d	10	ـُ	(damma)	u	39
ذ	ḏ	D	11	ـّ	(tašdīd)	w	40
ر	r	r	12	ـْ	(sukūn)	o	41
ز	z	z	13	ـَٔ	(ʒ-fatḥa)	^a	42
ژ	ž	^Z	14	ـٕ	(ʒ-kasra)	^i	43
س	s	s	15	ـُٔ	(ʒ-damma)	^u	44
ش	š	S	16	ـِّ	(tašdīd-kasra)	^w	45
ص	ṣ	^s	17	ء	ʒ	^_,^o	46
ض	ḍ	^d	18	ـ	(line segment)	_	47
ط	ṭ	^t	19	أ	ʒa	^A	48
ظ	ẓ	^z	20	إ	ʒi	^I	49
ع	ɛ	y	21	ؤ	ʒu	^U	50
غ	ġ	R	22	آ	(alif-madda)	E	51
ف	f	f	23	لآ	(lām-alif-madda)	^l	52
ق	q	q	24	ٱ	(upper wesla)	C	53
ڤ	v	v	25	لأ	(lām-alif-ʒ)	^L	54
ك	k	k	26	ي	(variant yāʒ)	Y	55
گ	g	g	27	ٱ	(lower wesla)	^C	56
ڭ	ṅ	^n	28	ـَّ	(tašdīd-fatḥa)	^e	57
ل	l	l	29				

Appendix A.

In the first column, the APO (Arabic–Persian–Ottoman) character is represented in its isolated form. Character ڭ comes from CARL FAULMANNs *Das Buch der Scrift, enthaltend die Schriftzeichen und Alphabete aller Zeiten und aller Völker des Erdkreises*, Vienna 1880, p. 104, "Türkische Nesχi", where it is transliterated as ṅ. The transliteration in column two comes from ARNE A. AMBROS's *Einführung in die moderne arabische Schriftsprache.* The input code of column three tries to be as short and as phonetic as possible; for example j stands for خ because of its sound in spanish, and y for ع because of its correspondence to the cyrillic ы which has almost the same phonetic value. Unfortunately, this correspondence cannot always be achieved: ة has no phonetic relation to c, etc. In the case of hamza, ^o is used for the "standing-alone hamza" ء and ^_ for the "hamza with carrier" ؤ (only initial and middle forms). Finally, in the fourth column you have a number which is purely inherent to yarbtex and will help you change the transliteration data in yarbtex.dat.

ألأرنب و الفيل

زعموا أن أرضاً من أرض الفيلة ، تتابعت عليها السنون و أجدبت ، فقلّ الماء في تلك البلاد و غارت العيون ، و أصاب الفيلة عطش شديد . فشكت ذلك إلى ملكها . فأرسل الملك رسله ورواده في التماس الماء في كل ناحيةٍ . فرجع إلية بعض رسله فأخبروه بأنهم و جدو في بعض الأمكنة عيناً تدعى القمرية ، كثيرة الماء . فتوجه ملك الفيلة بفيلته إلى تلك العين ليشربن منها . و كانت تلك الأرض أرض أرانب . فوطئت الفيلة الأرانب بأرجلها في جحرتها فأهلكن اكثرها . فاجتمع البقية منها إلى ملكها فقلن له : قد علمت ما أصابنا من الفيلة ، فاحتل لنا قبل رجوعهن علينا ؛ فإنهن راجعات لوردهن و مفنياتنا عن آخرنا . فقال ملكهن : ليحضرني كل ذي رأيٍ برأيه . فتقدم خزر منها يقال له فيروز ، وقد كان الملك عرفه بالأدن و الرأي ، فقال : إن رأى الملك أن يبعثني إلى الفيلة و يبعث معي أميناً يرى و يسمع ما أقول و ما اصنع و يخبره به ، فليفعل . فقال له ملك الأرانب : أنت أميني ، و أنا أرضى رأيك ، و اصدق قولك ؛ فانطلق إلى الفيلة و بلغ عني ما أحببت ، و اعمل برأيك ، و اعلم أن الرسول ، به وبرأيه و أدبه يعتبر عقل المر سل و كثير من شأنه ، و عليك باللين و المواتاة ؛ فإن الرسول هو يلين القلب إذا رفق ، و يخشن الصدر إذا خرق . فانطلق الأرنب في ليلةٍ ، القمر فيها طالع ، حتى انتهى إلى موضع الفيلة . فكره أن يدنو منهن فيطأنه بأرجلهن ، و إن لم يردن ذلك ، فأشرف على تلّ فنادى ملك الفيلة باسمه و قال له : إن القمر أرسلني إليك ، و الرسول مبلغ غير ملومٍ ، و إن أغلظ في القول . فقال له ملك الفيلة : و ما الرسالة ؟ قال : يقول لك القمر إنه من عرف فضل قوته على الضعفاء فاغتر بذلك من الأقوياء ، كانت قوته حينا و وبالاّ عليه ؛ و إنك قد عرفت فضل قوتك على الدواب فغرك ذلك مني فعمدت إلى عيني التي تسمى باسمي فشربت ماءها و كدرته أنت و أصحابك ؛ و إني أتقدم إليك و أنذرك ألا تأتيها فأعشي بصرك و أتلف نفسك . و إن كنت في شكٍ من رسالتي ، فهلم إلى العين من ساعتك ، فإني موافيك بها . فعجب ملك الفيلة من قول فيروز ، و انطلق معه إلى العين . فلما نظر إليها رأى ضوء القمر في الماء . فقال له فيروز : خذ بخرطومك من الماء و اغسل و جهك و اسجد القمر . ففعل . ولما أدخل خرطومه إلى الماء فحركه ، خيل إليه أن الماء يرتعد ، فقال ملك الفيلة : و ما شأن القمر يرتعد ؟ أتراه غضب من إدخال جحفلتي في الماء ؟ قال : نعم ، فاسجد له . فسجد الفيل للقمر و تاب إليه مما صنع ، و شرط له ألا يعود هو و لا أحد من فيلته إلى العين .

Appendix B. The Rabbit and the Elephant, from Kalila and Dimna.

Environment for Translating METAFONT to PostScript

Shimon Yanai and Daniel M. Berry

Abstract

This paper describes a program, mf2ps, that translates a METAFONT font definition into a definition for the same font in the PostScript language. mf2ps is constructed out of the part of the METAFONT program that extracts the envelopes of the letters; these envelopes are converted into PostScript outlines.

1 Introduction

This paper describes a program, mf2ps, that takes from a METAFONT [10, 11] program for a font all the necessary information in order to create an equivalent PostScript [1] font definition. The program makes use of the front end of the METAFONT program to extract the envelopes of the letters to produce the PostScript outlines. What makes this process natural is that both METAFONT and PostScript make liberal use of Bézier curves to describe non-circular curves.

By producing this translator, it is hoped to be able to produce from METAFONT fonts PostScript outline fonts which are more compact than the bitmapped fonts produced by the METAFONT program. Certainly the outline fonts are more easily scaled to other magnifications and possibly even other design sizes than are bitmaps. Moreover, doing so makes fonts heretofore available only on TEX [9] and other DVI-based formatters, available on ditroff [8] and other formatters which have evolved, or have been designed, for use with PostScript printers. This paper, which is typeset by ditroff, uses a PostScript version of the logo font in order to print the word "METAFONT" in the same appearance as in TEX-generated documents. Moreover, these new PostScript outline fonts can be used in TEX also! One needs only the TEXPS [3] software.

The organization of this paper is as follows. Section 2 presents the background of this work. Section 3 explains the rationale behind building the translator and describes a previous attempt at writing the translator and an approach to avoid. The software engineering aspect of the translator is described also in Section 3. The details of the implementation are exposed in Section 4. Section 5 describes the operation of the program. Section 6 evaluates the results. Finally Section 7 describes improvements to the translator that are left for future work.

2 Background

Typesetter formatting systems such as TEX and ditroff use fonts as raw material. The formatters accept mixed text and commands as input and produce output, which, if sent to the laser printers or typesetters, yields formatted text printed on pages. The laser printers and typesetters use fonts, i.e., sets of printable patterns, one per character, in various representations in order to cause the desired characters to appear on the printed form. For some printers, bitmaps are used, with 1's representing inked dots and 0's representing non-inked dots. Other printers accept commands that cause drawing of the characters, the printer providing the inked dots according to the drawing commands. One such popular command language is PostScript, and its usual use is to specify the outline of the character with the interpreting printer filling in the outline with ink. One popular method of describing fonts is with the METAFONT language, in which declarative definitions of how to paint the characters are given in terms of pen path and pen shape. Another popular method is the same PostScript that many printers accept. The prime difference is that the METAFONT program translates the font definitions into bitmaps prior to sending the font to the printer while a PostScript printer translates the outlines into bitmaps at the time of printing. Interestingly, both the METAFONT language and the PostScript language use Bézier curves for describing the curves followed by the pen or the outlines. As usually configured these days, TEX uses bitmapped fonts in the Computer Modern family generated by METAFONT, and ditroff uses PostScript outline fonts supplied by Adobe.

The subsequent subsections delve deeper into these issues in order to be able to state the goal of this paper in the next section.

2.1 Fonts, design sizes, and magnifications. As mentioned, fonts are the raw material of typesetting. A font is a set of printable patterns, one for each character, that causes printing of that character in a particular recognizable style on the page. As mentioned, these patterns can be represented by bitmaps or drawing instructions.

Characters come in various sizes. There are two independent notions of sizing for fonts, point size or design size and magnification. The *design size* is the size at which the character is designed to be used and is, in well-designed text, the size in which the character appears in final, printed copy. Design size is usually expressed in units of points, which are each approximately 1/72 of an inch. Most normal text in books, newspapers, and magazines is printed in 10 point type. Headlines are larger, perhaps as large as 30 points. The *magnification* of a font is the inverse of the ratio between the design size of the character and the size of the character as it emerges on the printer, the assumption being that the final copy is a photo reduction of the printed copy. Thus, if photo reduction halves linear dimensions, one prints with magnification 2. If everything is done right, then after reduction, the letter appears at its design size.

A 10 point design sized font printed at magnification 2 is similar to but not quite the same as a 20 point version of the same font. For example, the serifs on a large point size are smaller than they would be if strict linear magnification were used. Other proportions, e.g., of x-height to cap-height and of width to height, are also different. While many purists, Knuth included, insist on using a different pattern for each design size, many people accept magnification as yielding acceptable fonts at other point sizes. If the unit of magnification is not too big the results are acceptable even to many purists.

2.2 Problems with bitmapped fonts. A bitmap for a character is a rectangular array of bits covering the so-called bounding box or frame that exactly contains a letter. Figure 1 shows a low resolution bit map for the letter "N" in a sans serif font. The inked squares or pixels are denoted by "1" bits and the uninked pixels are denoted by "0" bits.

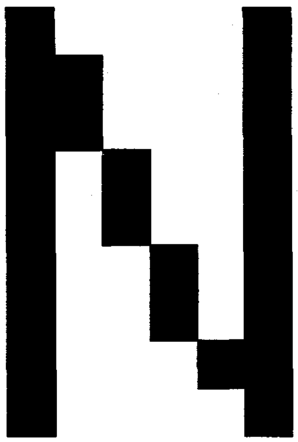


Figure 1

The low resolution example of Figure 1 illustrates a major problem with bitmapped fonts. Curved lines and straight lines that are neither vertical nor horizontal cannot be represented exactly by a rectangular pattern of pixels. One is forced to approximate them with rectangular steps. At high resolution, e.g. above 1000 or so, the human eye cannot see the steps, but at low resolution the steps are quite apparent. Visible steps are called "jaggies" after the jagged edges.

Bitmaps for a font must be built for each design size, magnification, and resolution. If the resolution is fixed, as is the case on most printers, a bitmap must be built for each design size and magnification. An attempt to use a given bitmap at a larger design size or magnification by just enlarging the area of each dot yields a bad case of jaggies.

2.3 METAFONT and its environment. METAFONT, a language for the specification of fonts or typefaces, has been used to provide fonts for the TEX family of typesetting systems. A METAFONT user writes a program for each letter or symbol of an alphabet. These programs are different from the usual computer programs, because they are essentially declarative rather than imperative, using an algebraic language to describe the center stroke or edges of the characters. The description of a letter in METAFONT is a set of equations describing the strokes. When combined with parameters describing the pen shape and size, one gets a full description of a letter. Sizes and shapes of pen nibs can be varied in METAFONT and the characters can be built up in such a way that the outlines of each stroke are precisely controlled. Herein lies the advantage of METAFONT; a font is easily specified and variations are obtained by varying parameters.

Currently, the program that converts a set of METAFONT font descriptions into a bitmapped font translates the description of a letter combined with a point size and a magnification into a bitmap. This bitmap can be sent to the printer to get a letter on the page. Herein lies a disadvantage of METAFONT; a bit map must be kept for each point size and magnification, and this can require a lot of space.

2.4 The POSTSCRIPT language. The POSTSCRIPT language is an interpretive programming language with graphics capabilities. POSTSCRIPT's extensive page description capabilities are embedded into a general-purpose programming language framework. The language includes a conventional set of data types such as numbers, arrays, and strings, control primitives such as conditionals, loops and procedures, and some unusual features such as dictionaries. In most POSTSCRIPT fonts, each letter is described by an imperative program tracing the outline of the letter. This tracing may include curves given as Bézier curves, straight lines, arcs, etc. A POSTSCRIPT printer interprets this outline program to draw and fill in the letters on the page. Some consider the imperative nature of POSTSCRIPT to be a disadvantage in comparison to META-

FONT's declarative nature. The main advantage of POSTSCRIPT relative to METAFONT is that one needs to keep only the outline. If, as in the usual case, the outline is specified in terms of a fixed path through Euclidean two-space, this outline may be scaled arbitrarily to yield any magnification. The scaling is done by the POSTSCRIPT interpreter at the printer. Thus the different magnifications do not require any additional storage space. Actually, the outlines are kept as if they were for the Adobe-standard 1000 dots per emm, which at a design size of 10 points amounts to 7200 dpi. Because a typical phototypesetter has a maximum resolution of about 2500 dpi, the outlines are said to be arbitrarily scaleable. If the outlines are kept, as are many METAFONT definitions, as paths through points calculated by the outline program, then it is possible to, say, make serifs grow more slowly than linearly. It would then be possible to have one POSTSCRIPT font scaleable to all design sizes. Generally, outline fonts are not written this way, so that strictly speaking they are scaleable only to all magnifications.

In addition, the POSTSCRIPT language has a way to work with bitmapped fonts. While the POSTSCRIPT printer can scale them before printing, the end result is that each of the fixed number of dots in the bitmap is made larger or smaller. Since the human will see larger dots as jagged lines, such fonts are not really considered scaleable.

2.5 Bézier curves. Both METAFONT and POSTSCRIPT use Bézier cubics to specify curves. For the Bézier form, four points are used, the start point, the end point, and two control points, as shown in the top half of Figure 2. The tangent vectors of the endpoints are determined from the line segments P_1P_2 and P_3P_4. The mathematical introduction of the Bézier form when given four points P_1, P_2, P_3, and P_4 is

$$z(t) = (1-t)^3P_1 + 3t(t-1)^2P_2 + 3t^2(1-t)P_3 + t^3P_4,$$

for $0 \le t \le 1$.

Two characteristics of the Bézier form tend to make it widely used in graphics. First, by choosing the control points one can easily mold the curve to a desired shape. Second, the four control points taken in another order define a convex polygon, $P_1\ P_2\ P_4\ P_3\ P_1$ in this case, the *convex hull*, which bounds the Bézier curve. The convex hull is useful in clipping a curve against a window.

When a METAFONT user specifies a path, METAFONT creates a list of knots and control points for the associated cubic spline curves. If the user has not specified the control points explicitly, METAFONT itself finds some for the splines of a curve, while POSTSCRIPT requires all the four points to be explicitly given.

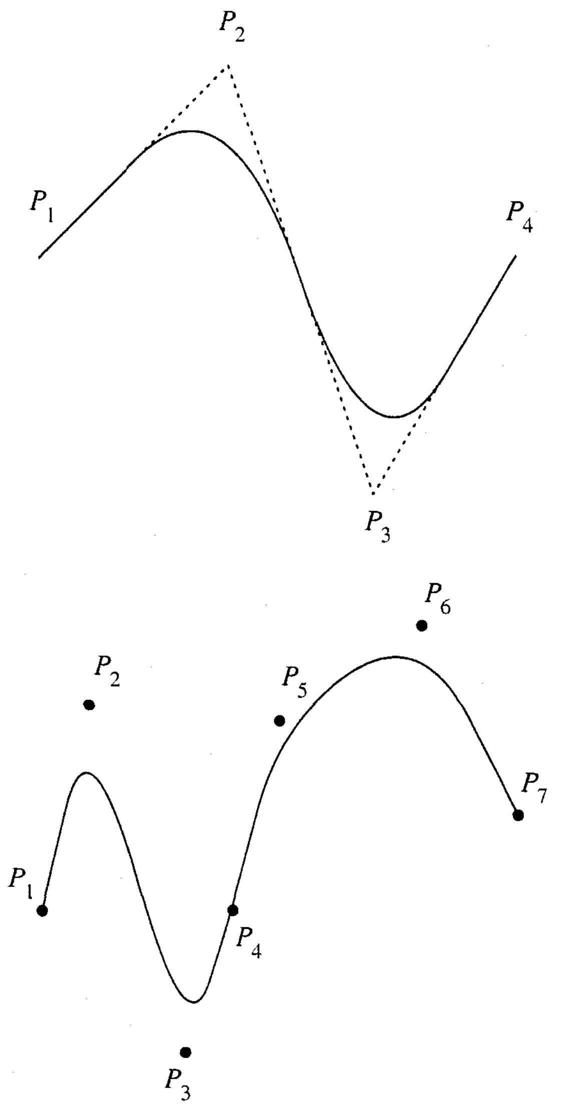


Figure 2

3 METAFONT to POSTSCRIPT compiler—why and how

This section describes a major performance problem with METAFONT-generated fonts that perhaps can be solved by translating them into POSTSCRIPT fonts. The goals of this translation are established. Based on these goals, a particular approach is adopted to engineer the software largely from existing components.

3.1 A problem with METAFONT-generated bitmapped fonts. In METAFONT, one gets one bitmap per point size and magnification. The size of these bitmaps grows as the square of product of the design size and the magnification and requires a large storage space. Files that are sent to the printer will be large, especially if lots of different point sizes or magnifications are used. In POSTSCRIPT with outline fonts, there is one outline per character which can be scaled arbitrarily to any magnification that might be needed. Moreover, POSTSCRIPT outline fonts are generally more compact than bitmapped fonts. For example, an enclosed rectangle is represented by its four corner points rather than by all the bits enclosed by the rectangle.

Certainly the outline fonts are more easily scaled to other magnifications. By scaling the bitmapped fonts downward, too much information is lost, and scaling upward introduces the jaggies. Moreover, the pixel array is device dependent; it is valid for output devices of only one particular resolution and one choice of possible data values per pixel. Scaleable fonts have a great advantage — you need only one font description file for all magnifications of that font. Actually, POSTSCRIPT outline fonts are more scaleable even than the METAFONT originals for another reason. In [9], it is said, "Caution: before using this '**at**' feature (i.e. scaling downward or upward) you should check to make sure that your typesetter supports the font at the size in question; TEX will accept any ⟨desired size⟩ that is positive and less than 2048 points, but the final output will not be right unless the scaled font really is available on your printing device." Getting POSTSCRIPT outline versions of METAFONT fonts is possible since both are based on Bézier curves. Doing so makes fonts heretofore available only on TEX and other DVI-based formatters available on ditroff and other formatters which have evolved to or have been designed for use with POSTSCRIPT printers.

3.2 Goals. Based on the observations of Section 3.1, the goal of this research is to produce a METAFONT to POSTSCRIPT compiler, mf2ps. Its operational requirements are items 1 through 5:

1. It must be possible to translate any legitimate METAFONT font definition at any given design size into a POSTSCRIPT outline font.
2. The resulting POSTSCRIPT outline font should be arbitrarily scaleable.
3. The resulting fonts should look like the bitmapped fonts when printed on the same printer.
4. The resulting POSTSCRIPT outline font should be more compact *when sent to the printer* than a POSTSCRIPT version of the METAFONT-generated bitmapped font.

The fourth requirement deserves a bit of explanation and qualification. First note that what is compared is what is sent to the printer. Certainly there are compressed versions of the bitmapped fonts that reduce the disk storage requirements of the bitmapped fonts. However, they must be uncompressed before sending them to most printers. It is the printer's storage that is limited; generally disk space is in abundance. However, since printers these days are general purpose computers, what a printer accepts may in fact be a compression that it has been programmed to undo.

Now for the case in which disk space is of concern, the comparison should still be relative to printable versions. There exist algorithms, e.g. that of Lempel and Ziv [13] that can be used to compress POSTSCRIPT outline fonts which are, after all, just ASCII files. Therefore, in order not to have a contest between compression algorithms, the uncompressed versions are compared. Furthermore, in order not to have a contest between different kinds of printers that may have differing font representations, POSTSCRIPT outline fonts are compared to POSTSCRIPT bitmapped fonts. When considering disk space, the fact that one bitmapped font is needed for each magnification is taken into account. Thus, the interest is in comparing the size of a scaleable outline font to the total storage for the bitmapped fonts for all magnifications of a given design size.

5. The resulting POSTSCRIPT outline font should be more compact than the total of the sizes of the POSTSCRIPT versions of the METAFONT-generated bitmapped fonts at each available magnification. Even this comparison is not completely fair since only specific magnifications are provided, while the POSTSCRIPT font is arbitrarily scaleable.

Observe finally, that the comparison is against magnifications of a single design size since purists would argue that there should be a different outline font for each design size. Since there are those that do not require this purity, the various design sizes will be compared also.

The software engineering goal is item 6.

6. mf2ps should be written as much as possible using the existing METAFONT program both to save work and to ensure that all METAFONT-acceptable font definitions are handled.

The evaluation of the results will be done relative to these goals.

3.3 Previous attempts. Leslie Carr wrote a collection of programs to produce POSTSCRIPT outline fonts from METAFONT fonts in 1987. Carr's programs take as input the *log* output file of METAFONT which contains a description of all the paths that METAFONT traces out in drawing a character.

Carr has problems of information loss as a result of not having entered into the METAFONT program. This is the reason why Carr's characters are poor looking. In [5], Carr observes, "In the cmr10 font, the *crisp* pen has diameter zero, so serifs have square corners. In the cmtt10 font, *crisp* is set to a larger value and the serifs end in semicircles. Because the shape of the current pen can NOT be taken into account in POSTSCRIPT, these differences in the characters shapes will not be seen. This is a **fundamental** problem: given a path p and a pen q (whose shape is also an arbitrary path), METAFONT effectively envelopes p with respect to the shape of q; POSTSCRIPT can do nothing other than stroke it to produce a line of constant width. This incompatibility comes to light when the width of the pen is significant to the shape of the character".

In order to avoid this problem, mf2ps finds the internally generated envelope, which is used as the boundaries of the inked region, and uses this envelope as the outline. It does not matter, then, what the pen path and the pen shape are.

More recently, during the time that the work described herein was being done, there were other efforts with similar goals.

Doug Henderson [6] obtained outline font characters by modifying the endchar macro, which is called for each character after the bitmap is generated, to take the bitmap for the character and white out all but the bits on the edge. The number of bits left on the edge is varied according to the resolution of the bitmap. These outlines, being bitmapped, are just as unscaleable as are the bitmaps for the filled-in characters.

Neil Raine and Graham Toal [12] have developed software that takes the bitmaps and rediscovers the outlines by tracing the pixels. The outlines that are used as the basis for POSTSCRIPT fonts are, for the most part, generated from bitmaps at 2400 dpi. They first generate RISC OS outline fonts which are screen fonts for Acorn's Archimedes RISC computer. These are true scaleable outlines. Then, these outlines are converted into POSTSCRIPT format. Toal says that the the quality of the fonts produced is not too great at low resolutions because of shortcomings in Adobe's rendering algorithm. He adds that at 1200 dpi on a phototypesetter, they are indistinguishable from METAFONT-generated bitmapped fonts. These authors suspect that information that is critical for good appearance is lost when tracing an outline on a bitmap generated from a mathematically described envelope. Better results should be obtainable using the original envelope.

John Hobby [7] has developed a program called MetaPost, which translates from an extension of METAFONT into POSTSCRIPT cubic splines and commands. His goal was to turn METAFONT into a system for typesetting general graphics, including embedded text. His approach, similar to ours, was to modify the METAFONT program into what he desired. Befitting his more general goals, besides modifying the output, he has added new commands to the input language. Moreover, his translation appears to be a direct mapping from a METAFONT command sequence to a POSTSCRIPT command sequence. The result is a program more powerful than mf2ps. It will be interesting to compare fonts produced by MetaPost and mf2ps for appearance and performance.

3.4 Methodology. There are a number of ways to build the compiler. They include

1. writing the whole compiler from METAFONT to POSTSCRIPT from scratch: This has the advantage that one does not have to get into another person's software, which is not very pleasant when the software is so big. On the other hand, one would have to treat the whole job of turning mathematical equations and any arbitrary pen shape into outlines.
2. using the METAFONT output as was done by Leslie Carr [5]: This has the advantage of not requiring delving into another's software, but the generated information is not enough if one wants no deviations from the originals.
3. getting into the METAFONT program: This requires examining the internals of the METAFONT program. However, METAFONT and POSTSCRIPT make liberal use of Bézier curves to describe noncircular curves. This fact makes the translation process natural. For each specified path, METAFONT creates control points for the associated cubic spline curves before calculating the bit map. METAFONT also calculates the edge offsets implied by the pen shape. Using the necessary information one can get a new set of control points that define Bézier curves and lines that are needed to build the POSTSCRIPT outline fonts.

3.5 Software engineering of solution. The idea is to split the METAFONT program into front end and back end. The front end takes METAFONT specification of a character, magnification, and point size, and produces the envelope, i.e., the outline of the character, and the back end fills the envelope with bits. Taking the existing front end and writing a new back end that converts the envelope into a POSTSCRIPT specification of an outline is our method of producing mf2ps. The bit-filling process will be done by the printer.

In order to make POSTSCRIPT fonts arbitrarily scaleable, we have to ask the mf2ps program to use a very large magnification, at least to try to match the grid on which Adobe plots the points of its outlines. Adobe plots its characters on a 1000×1000 grid. Thus, Adobe's resolution is 1000 dpm (dots per em), which for design size 10 points is 7200 dpi. Unfortunately, METAFONT, and thus mf2ps accepts resolutions only up to 3000 dpi. The results should be sufficient to produce fonts scaleable up to magnification 7 or 8, which is a reasonable range in typesetting.

This approach helps meet goal 6 because the original unchanged METAFONT program is used. Thus, exactly the same input is accepted as in the METAFONT program. There is some extra frosting obtained by the chosen approach. The program for translating METAFONT to POSTSCRIPT is actually a bit of an interactive environment because the new back end is an extension of the existing one. This existing back-end provides an interpreter that executes a METAFONT character definition and displays the defined character on the screen. Figure 3 shows the dump of a screen containing several windows, one showing a METAFONT definition, another showing the result of its interpretation, and a third containing the POSTSCRIPT translation of the definition in the first window. If software to interpret POSTSCRIPT definitions were available here, a fourth window could be set up showing the result of interpreting the translation of the third window. This would allow comparison of the character's appearances without having to print them on paper.

4 The program

In the following discussion, the METAFONT program is often called just "METAFONT".

The METAFONT program has been written so that it can be made to run efficiently in a wide variety of operating environments by making comparatively few changes. Such flexibility is possible because the program is written in the WEB language which is at a higher level than Pascal. The preprocessing step that converts WEB to Pascal is able to introduce most of the necessary refinements. Semiautomatic translation to other languages is also feasible, because the program does not make extensive use of features that are peculiar to Pascal.

The program has two important variations: First, there is a long and slow version called INIMF, which does the extra calculations needed to initialize METAFONT's internal tables. It has to be run first. It initializes everything from scratch without reading a base file, and it has the capability of dumping a base file. Secondly, there is a shorter and faster production version called VIRMF, which cuts the initialization to a bare minimum. It is a virgin program that needs to input a base file in order to get started. VIRMF typically has more memory capacity than INIMF, because it does not need the space consumed by the dumping and undumping routines, etc.

In order to generate a compiler that translates METAFONT to POSTSCRIPT, additional external procedures and functions were added to the METAFONT program so that it runs exactly the same except that when it asks for an output file name, it asks for an additional name, for the extra output file that is to contain the POSTSCRIPT outlines. Those changes were made on the Pascal version of the VIRMF, and were compiled later with METAFONT's library files. (It was a complete oversight on our part not to have modified the WEB version of VIRMF.) A few extra lines were added to the macro file, `plain.mf`. These act as flags, identifying that METAFONT has entered some of the macros.

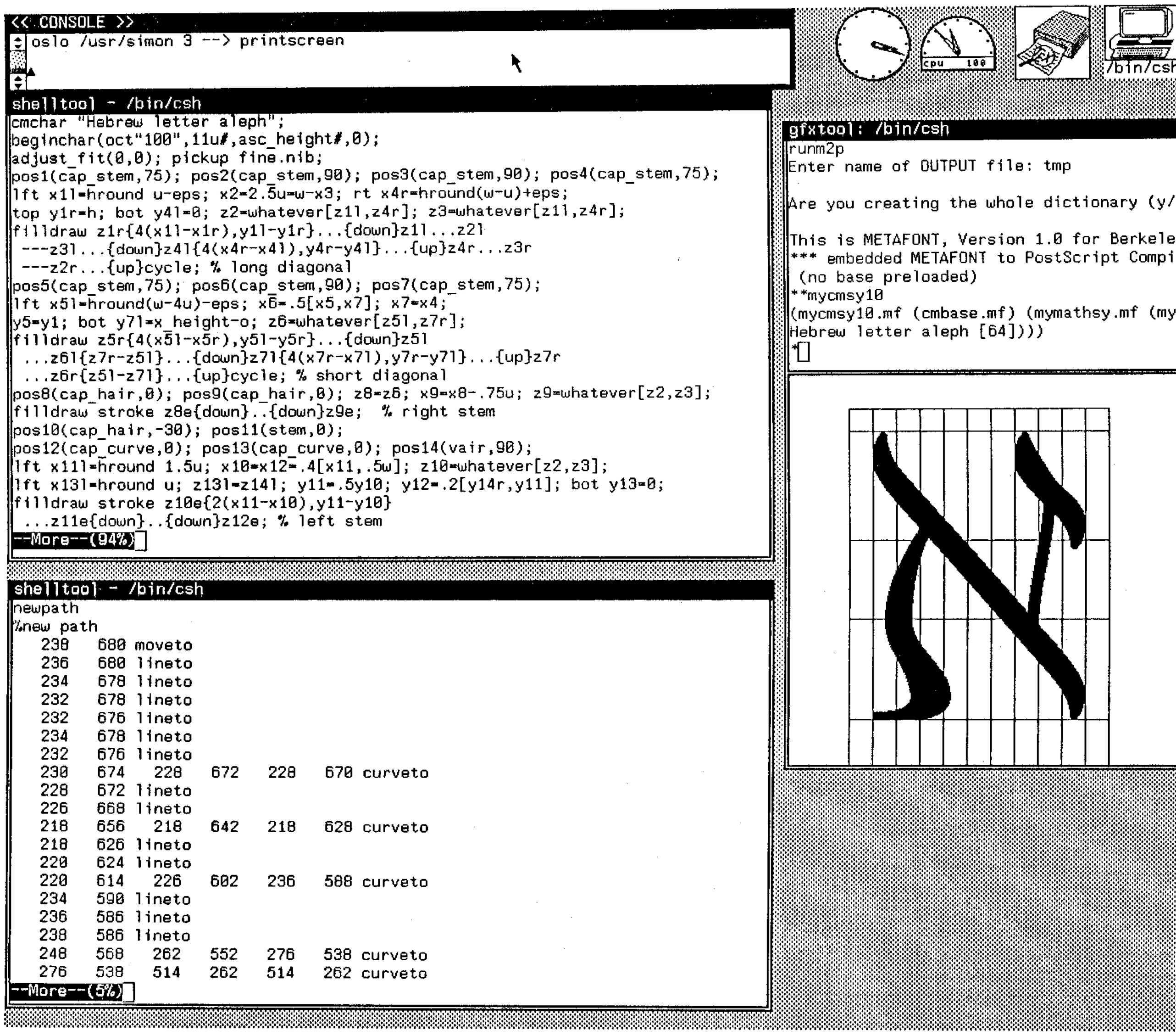
4.1 Basic idea. To specify a character in METAFONT, one specifies either an envelope (outline) or a center-line path and a pen head. For the former, METAFONT just fills the envelope with bits. For the latter, METAFONT pretends that it is drawing the character with a pen of specified head shape following the specified path, i.e., the center of the head stays on the path. The distance from the center-line path and outer edge of ink trail left by pen head is called the *offset*. So, for a character, METAFONT follows the center-line path to calculate the path of offset points, i.e., the envelope, and then fills the envelope with bits. In either case, METAFONT ends up filling an envelope.

We need to break METAFONT into a front end and a back end at the point just after the envelope has been calculated. Then we provide a new back end that converts the envelope into POSTSCRIPT instead of filling the envelope with bits. Note then that the POSTSCRIPT printer will fill in the envelope with bits as it fills the path obtained from the envelope.

The following subsections describe the data and the calculations involved in the new back end.

4.2 Data structures. The main data structures that METAFONT keeps for a character are the center-line path, the pen shape, and the envelope path. There are a few operations that can be performed on paths, called transformations.

4.2.1 METAFONT's path representation. When a METAFONT user specifies a path, METAFONT creates a list of knots and control points for the associated cubic



```
<< CONSOLE >>
oslo /usr/simon 3 --> printscreen
```

```
shelltool - /bin/csh
cmchar "Hebrew letter aleph";
beginchar(oct"100",11u#,asc_height#,0);
adjust_fit(0,0); pickup fine.nib;
pos1(cap_stem,75); pos2(cap_stem,90); pos3(cap_stem,90); pos4(cap_stem,75);
lft x1l=hround u-eps; x2=2.5u=w-x3; rt x4r=hround(w-u)+eps;
top y1r=h; bot y4l=0; z2=whatever[z1l,z4r]; z3=whatever[z1l,z4r];
filldraw z1r{4(x1l-x1r),y1l-y1r}...{down}z1l...z2l
 ---z3l...{down}z4l{4(x4r-x4l),y4r-y4l}...{up}z4r...z3r
 ---z2r...{up}cycle; % long diagonal
pos5(cap_stem,75); pos6(cap_stem,90); pos7(cap_stem,75);
lft x5l=hround(w-4u)-eps; x6=.5[x5,x7]; x7=x4;
y5=y1; bot y7l=x_height-o; z6=whatever[z5l,z7r];
filldraw z5r{4(x5l-x5r),y5l-y5r}...{down}z5l
 ...z6l{z7r-z5l}...{down}z7l{4(x7r-x7l),y7r-y7l}...{up}z7r
 ...z6r{z5l-z7l}...{up}cycle; % short diagonal
pos8(cap_hair,0); pos9(cap_hair,0); z8=z6; x9=x8-.75u; z9=whatever[z2,z3];
filldraw stroke z8e{down}..{down}z9e;  % right stem
pos10(cap_hair,-30); pos11(stem,0);
pos12(cap_curve,0); pos13(cap_curve,0); pos14(vair,90);
lft x11l=hround 1.5u; x10=x12=.4[x11,.5w]; z10=whatever[z2,z3];
lft x13l=hround u; z13l=z14l; y11=.5y10; y12=.2[y14r,y11]; bot y13=0;
filldraw stroke z10e{2(x11-x10),y11-y10}
 ...z11e{down}..{down}z12e; % left stem
--More--(94%)
```

```
shelltool - /bin/csh
newpath
%new path
   230   680 moveto
   236   680 lineto
   234   678 lineto
   232   678 lineto
   232   676 lineto
   234   678 lineto
   232   676 lineto
   230   674   228   672   228   670 curveto
   228   672 lineto
   226   668 lineto
   218   656   218   642   218   628 curveto
   218   626 lineto
   220   624 lineto
   220   614   226   602   236   588 curveto
   234   590 lineto
   236   586 lineto
   238   586 lineto
   248   568   262   552   276   538 curveto
   276   538   514   262   514   262 curveto
--More--(5%)
```

```
gfxtool: /bin/csh
runm2p
Enter name of OUTPUT file: tmp

Are you creating the whole dictionary (y/

This is METAFONT, Version 1.0 for Berkele
*** embedded METAFONT to PostScript Compi
 (no base preloaded)
**mycmsy10
(mycmsy10.mf (cmbase.mf) (mymathsy.mf (my
Hebrew letter aleph [64])))
*
```

Figure 3

spline curves. If the knots are $z_0, z_1, \ldots, z_n$, there are control points z_k^+ and z_{k+1}^- such that the cubic splines between the knots z_k and z_{k+1} are defined by the Bézier formula

$$\begin{aligned} z(t) &= B(z_k, z_k^+, z_{k+1}^-, z_{k+1}; t) \\ &= (1-t)^3 z_k + 3t(t-1)^2 z_k^+ \\ &\qquad + 3t^2(1-t) z_{k+1}^- + t^3 z_{k+1}, \end{aligned}$$

for $0 \le t \le 1$.

There is a 7-word node for each knot z_k, containing one word of control information and six words for the x and y coordinates of z_k^- and z_k and z_k^+. The control information appears in the *left_type* and *right_type* fields and they specify properties of the curve as it enters and leaves the knot. There is also a *link* field, which points to the following knot. Before the Bézier control points have been calculated, the memory space they will ultimately occupy is taken up by information that can be used to compute them. The METAFONT *make_choices* procedure chooses angles and control points for the splines of a curve when the user has not specified them explicitly.

4.2.2 METAFONT's path transformation. When METAFONT digitizes a path, it reduces the problem to the special case of paths that travel in the *first octant* directions; i.e., each cubic $z(t) = (x(t), y(t))$ being digitized will have the property that $0 \le y'(t) \le x'(t)$. This assumption makes digitizing simpler and faster than if the direction of motion has to be tested repeatedly. When $z(t)$ is cubic, $x'(t)$ and $y'(t)$ are quadratic, hence each of the four polynomials, $x'(t)$, $y'(t)$, $x'(t)-y'(t)$, and $x'(t)+y'(t)$, crosses through 0 at most twice. If we subdivide the given cubic at these places, we get at most nine subintervals. In each of these intervals each of $x'(t)$, $y'(t)$, $x'(t)-y'(t)$, and $x'(t)+y'(t)$ has a constant sign. The curve can be transformed in each of these subintervals so that it travels entirely in first octant directions, if we exchange x and $-x$, y and $-y$, and x and y as necessary.

4.3 Pens and envelopes. There are two kinds of pen heads that may be used, polygonal and elliptic. There are a number of trade-offs involved in their use. The first subsection treats the case of an n-vertex polygonal pen shape and the second treats the case of an elliptical pen shape. Both describe the influence of pen shape on the envelope of the font.

4.3.1 Polygonal pens. Suppose that the vertices of a polygon are $w_0, w_1, \ldots, w_{n-1}, w_n = w_0$ in counterclockwise order. A convexity condition requires that each vertex turns left when one proceeds from w_0 to $w_1 \cdots$ to w_n. The envelope is obtained if we offset a given curve $z(t)$ by w_k when that curve is traveling in a direction $z'(t)$ lying between the directions $w_k - w_{k-1}$ and $w_{k+1} - w_k$. At times t when the curve direction $z'(t)$ increases past $w_{k+1} - w_k$, METAFONT temporarily stops plotting the offset curve and inserts a straight line from $z(t)+w_k$ to $z(t)+w_{k+1}$; notice that this straight line is tangent to the offset curve. Similarly, when the curve direction decreases past $w_k - w_{k-1}$, METAFONT stops plotting and inserts a straight line from $z(t)+w_k$ to $z(t)+w_{k-1}$; the latter line is actually a retrograde step, which will not be part of the final envelope under METAFONT's assumptions. The result of this consideration is a continuous path that consists of alternating curves and straight line segments. The segments are usually so short, in practice, that they blend with the curves.

4.3.2 Elliptical pens. To get the envelope of a cyclic path with respect to an ellipse, METAFONT calculates the envelope with respect to a polygonal approximation to the ellipse. This has two important advantages over trying to obtain the exact envelope:

1. Polygonal envelopes give better results, because the polygon has been designed to counteract problems that arise from digitization; the polygon includes sub-pixel corrections to an exact ellipse that make the results essentially independent of where the path falls on the raster.
2. Polygonal envelopes of cubic splines are cubic splines. Hence it is not necessary to introduce completely different routines. By contrast, exact envelopes of cubic splines with respect to ellipses are complicated curves, more difficult to plot than cubics.

4.4 Taking out data. After METAFONT has calculated the paths and the offsets, it is ready to send the values to the *make_moves* procedure which generates discrete moves for any four points that represent a Bézier curve. This is done for each one of the cyclic paths from which the letter is built. When the offsets are zero, this is done by the *fill_spec* procedure. Otherwise this is done by the *fill_envelope* procedure. In the latter case, the line segments, which were discussed earlier, should be taken out also in order to get smooth connections between the different curves that the cyclic path is built from. Because POSTSCRIPT describes any shape in terms of curves and lines, this is the point to take advantage of METAFONT's calculations, i.e., when METAFONT calls the *make_moves* procedure and when METAFONT draws line segments for offset corrections.

4.5 Processing the data. The generated data are not ready yet to be used. First, we should unskew, i.e., transform from the first octant back to the original, the paths according to the octant that the paths were traveled in before they were skewed. This unskewing is done by taking out the octant number at the moment that the *make_moves* procedure is called and then using METAFONT's *unskew* procedure that sets values x' and y' to the original coordinate values of a point, given an octant code and coordinates (x,y) after they have been mapped into the first octant and skewed; the new values are sent to the *send_p_s* procedure. This procedure has eight formal parameters that are all used when sending a curve. When sending a line, only four parameters are used, two to denote the start point and two to denote the end point; the remaining four parameters are sent as zeros so *send_p_s* can distinguish whether a line was sent or a curve. In the next step, *send_p_s* unscales the numbers because METAFONT works with units of scaled points, of which there are 2^{16} in an ordinary point. While unscaling, the values are transformed in order to send them to the POSTSCRIPT dictionary `FontBBox` command. After this pre-processing, the data are sent to a temporary file.

4.5.1 Getting more information. When METAFONT calls the *make_moves* procedure, it does not have any information on the role that this path is going to play, whether the current cyclic path is going to be *filled* or whether it will act as a boundary of a region to be *erased*.

In order to distinguish between the cases, more information has to be taken. This is done by copying the `plain.mf` file into a new file named `myplain.mf` and adding a few lines to it. The additional code was added in order to identify METAFONT's use of the macros. METAFONT uses the variables for date only once, when the program is started, so it was decided to use them in the rest of the program. The `year` is changed to -1 when METAFONT's `pen_stroke` macro is applied on a cyclic path, i.e., in the characters such as "o", "O", and "Q", and to -2 when the `erase` macro is called. The `month` is changed when the `fill` macro is called. There are three kinds of paths:

1. paths to be *filled* are processed using the POSTSCRIPT `fill` command.
2. paths to be *stroked* are processed using the POSTSCRIPT `eofill` command.
3. paths to be *erased* are processed using specialized procedures which will be discussed later.

A letter cannot always be treated as one unit by means of the `fill` and `eofill` commands. For instance, the letter "Q" is built of two different paths, the first of which is stroked and the second of which is filled. Generating the letter using the POSTSCRIPT `eofill` command causes a hole in the image (see Figure 4).

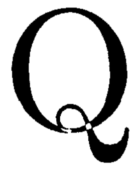


Figure 4

So while generating a letter, fill mode can be changed for each cyclic path. Moreover, when generating a letter whose paths should be filled, it is not always possible to use just one `fill` command (see Figure 5).

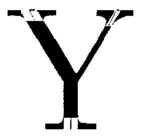


Figure 5

When a POSTSCRIPT `fill` command is applied to a path that is composed of more than one subpath, say two for the sake of simplicity, and one subpath is inside the other and is drawn in a direction opposite to the external one, the internal path is considered a hole and is not filled (see Figure 6). So, if several paths are to be filled in this manner, the description of each one of them should be ended with the `fill` command. There is one more benefit to using this strategy: The POSTSCRIPT `current path` stack becomes empty after encountering any kind of `fill` command. Therefore, using the `fill` command after each path can help avoid `stack overflow errors` if all paths together are too long.

4.5.2 Treating erasing paths. There are three methods of handling the problem of paths that should be erased by mf2ps itself:

1. filling with white: Because erasing paths are built in order to erase an existing filled area and POSTSCRIPT overlaps paths (i.e., a region is shown in the color that was drawn last), erasing paths can be implemented by filling those paths with white. This solution is the easiest, but it works only if the background is white and the letter is drawn in some level of gray. If one wants to draw a letter with background other than white, the resulting

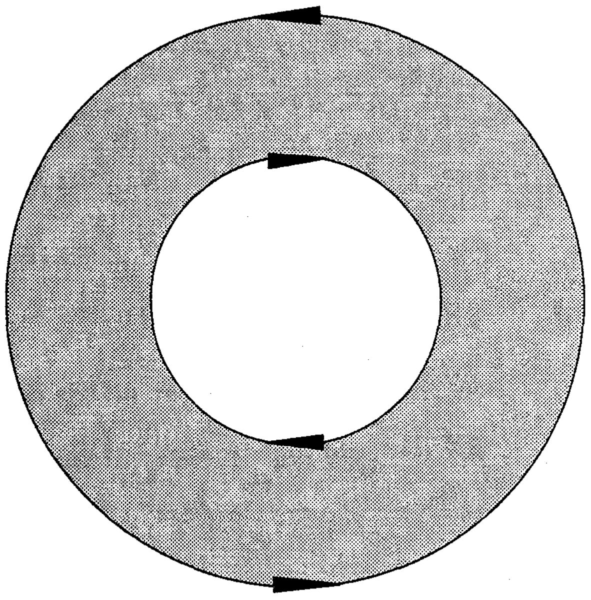


Figure 6

appearance will not be correct.

2. calculating new paths resulting from subtracting the erasing paths from the previous filled paths: Such a solution can be global. However, it costs a lot in terms of processing time and accuracy, because paths are given implicitly by four points, and in order to calculate the new paths, one should find the intersection points of Bézier curves, i.e., to find points that lie on both Bézier curves, and then calculate new curves, which are difficult to calculate from those points.

3. using the PostScript `eoclip` command: Because the letters are bounded in a 1000 ×1000 box, a primary square path whose segments are 1000 units long should be declared and after it all the erasing paths should be listed. After relocating the erasing paths we are ready to declare `eoclip`, which means that the clipping path is the external primary one and the internal paths, the erasing paths, are holes. This is an elegant solution that uses the power of the language and is available in simple situations in which there is no intersection between the erasing paths (see Figure 7). If there were intersections, a little more sophisticated use of the `eoclip` command would be needed. Relocation of the erasing paths is done by the procedure `doarrange`.

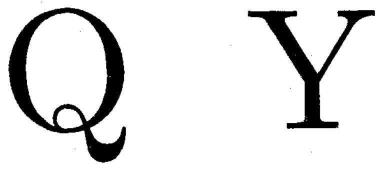


Figure 7

There are other problems caused by the erasing paths. Because the erasing paths have segments in common with paths to be filled, PostScript must decide whether the common segments are in the clipping path or not. PostScript does not seem to have a consistent policy on that and it seems to be that the decision is taken arbitrarily (see Figure 8).

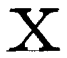


Figure 8

An attempt to resolve the clipping path problem led to the first author sending the following electronic message (obviously, not as nicely formatted as herein) to Glenn Reid of Adobe Systems, Inc.

From simon Tue Mar 21 13:22:32 1989
To: greid@adobe.com
Subject: Problem in PostScript

Dear Mr. Reid

I have got a problem in understanding the PostScript policy in determining "what is in the clipping path". I think there is a problem in the boundaries. Here is an example that shows that problem:

```
gsave
initclip
newpath

0 0 moveto
0 1000 lineto
1000 1000 lineto
1000 0 lineto
0 0 lineto

300 100 moveto
700 100 lineto
700 300 lineto
300 300 lineto
```

```
    300 100 lineto

    700 900 moveto
    300 900 lineto
    300 700 lineto
    700 700 lineto
    700 900 lineto

    eoclip

    newpath
    100 100 moveto
    900 100 lineto
    900 900 lineto
    100 900 lineto
    100 100 lineto
    fill
    grestore
```

As you see, the problem is that on top of the shape, the line which belongs to the upper "hole" in the clipping path and to the current path (to be filled) is drawn, and on bottom of the shape it is not.

This is happening both on the Apple Laser printer and on the QMS-80.

I would be glad to have a reply from you.

Thanks in advance
Shimon Yanai
C.S Dep.
Technion

What Mr. Reid saw when he printed the POSTSCRIPT commands contained in the message is reproduced in Figure 9.

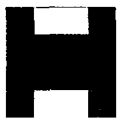


Figure 9

Mr. Reid replied with the following:

From: greid@adobe.com (Glenn Reid)
To: Shimon Yanai <simon@techunix>
Cc: greid@adobe.com
Subject: Re: Problem in PostScript
In-Reply-To: Your message of Wed, 22 Mar 89 ...
Date: Wed, 22 Mar 89 11:41:35 PST

The problem is that the path you are filling falls exactly on the edge of the clipping path. This produces a zero-width area to fill, and unfortunately it sometimes fills and sometimes does not with the current fill algorithm. I believe that it is related to the direction of the paths; if the paths are going in opposite directions along the same line, it will fill with a one-pixel area, but if they are going in the same direction, it will not fill. I believe this has been fixed to be more consistent in Display PostScript, for what it's worth.

Glenn Reid
Adobe Systems

The idea of using opposite directions had been checked before sending the letter, so the problem had to be solved within the back end of mf2ps. The erasing paths near the top of the letter had their y coordinates increased by 0.8 points, and those near the bottom had their y coordinates decreased by the same amount. This shift is invisible to the human eye because the font definitions are in terms of hundreds of points (see Figure 10). This solution was designed to work with most existing METAFONT fonts. It is possible that there will be fonts that are not treated well by this solution.

Figure 10

4.6 Optimization. Optimization is done in order to make the description of the fonts shorter and to save work in the POSTSCRIPT interpreter. This is done in three ways:

1. not printing lines with length zero. As was said earlier, the METAFONT program prints lines to connect offset points. There are times that after rounding or truncating the output data, the start point and the end point are equal. In such cases, the lines are eliminated.
2. checking if the Bézier curve acts as a line. From the definition of the Bézier curve, it is known that if the two control points lie on the line that connects the start point and the end point, the curve is of degree one. In such cases mf2ps generates a command to print a line from the start point to the end point, thus saving space and avoiding redundant calculations for the POSTSCRIPT interpreter.

3. checking if a series of consecutive line segments are in the same line. This is done by storing the segments in a buffer and checking whether a new segment is collinear with the last stored.

4.7 Changed or added routines. The following is a list of routines that were changed or added in order to build mf2ps from METAFONT.

printchar was modified to get character names.

fixdateandtime was modified to initialize variables that were used as flags in the macros.

fillspec was modified to send out data on splines.

skewlineedges was modified to send out offset lines.

dualmoves was modified to send out offset lines.

fillenvelope was modified to send out data on splines.

dostatement was modified to identify tokens that are strings.

main was modified to call the mf2ps procedure in the beginning and ending of the program.

sendcurve was added to unskew spline values and to send them to the next process.

sendline was added to unskew line values and to send them to the next process.

ok was added to check if two lines are collinear.

restore was added to restore the parameters of the last line.

recall was added to recall values from the buffer.

us was added to convert the METAFONT scale so that a letter would fit the Adobe standard 1000×1000 bounding box.

send_p_s was added to create a POSTSCRIPT file of lines and curves.

makemoves was modified to send out spline data.

dump was added to append information from the file named `f` to the file named `g`.

checkerase was added to identify the file that contains "erase" commands, and their position within the file.

doarrange was added to put erasing paths at the beginning of the file.

print_start was added to signal the beginning of a new cyclic path to be processed.

print_end was added to signal the end of the current cyclic path.

init_ps was added to make initializations.

makenewdef was added to make initializations when more than one character occurs in the input.

closeolddef was added to close the last definition.

tini_ps was added to handle the ending of the process.

auxprintchar was added to print characters.

auxprint was added to print strings.

5 Operation of mf2ps in a UNIX environment

When invoked, mf2ps first asks for an output file name. For the example this file is called `ex1`. mf2ps then asks,

```
"Are you creating the whole dic-
tionary (y/n)?".
```

If the answer is other than "`y`" or "`Y`", it is considered "`no`". If the answer is "`y`" or "`Y`", then the whole dictionary is created. This means that mf2ps creates a POSTSCRIPT dictionary that includes entries for all the characters that are in the input, e.g., `cmr10` set. This dictionary needs additional definitions such as *left side bearing, width, bounding box*, etc. These definitions need information on character features that must be calculated within the program. Otherwise, the whole dictionary is not created and the program treats the input as a single character definition that is to be translated into a POSTSCRIPT outline definition. After mf2ps prompts "`**`", we are in the METAFONT environment. Now the user inputs

```
\mode=hires;\nodisplays;\input cmr10;↵
```

After mf2ps has finished, the resulting POSTSCRIPT font dictionary can be used to print text. In order to print text, the font dictionary should be installed in some formatter's font source directory, and then it can be loaded through the formatter's commands. The dictionary followed by appropriate show and showpage commands can also be sent directly to the printer.

6 Evaluation of results

This section evaluates the mf2ps program relative to goals established in section 3.2. The program was produced as a variation of METAFONT and it accepts any METAFONT *font definition and produces a* POSTSCRIPT outline font scaleable up to magnification 8, or to point size 80 if you are not a purist. Thus goals 6 and 1 have been entirely met and goal 2 is partially met. To meet goal 2 fully the program must be modified to allow large enough arrays to handle magnifications up to 7200; this is left to future work.

It remains to evaluate the appearance and sizes of the outline fonts relative to the bitmapped fonts to see if goals 3, 4, and 5 have been met.

6.1 Appearance. In order to compare appearances, the outline font (Subsubfigure P) and and the 300 dpi bitmapped font (Subsubfigure M) generated from the same METAFONT definition are used to print similar sentences at one, two, or three different sizes or magnifications on three devices of differing resolutions. The sentences are printed in the `cmr` (Subfigure R), `cmtt` (Subfigure T), and `lasy` (Subfigure S) typefaces. The bitmapped fonts may be printed at design sizes 7, 8, 10, or 12, and the outline fonts may be printed at magnifications .7, .8, 1.0, or 1.2. Finally, the three devices are the 300 dpi LaserWriterII (Figure 11-LW300), the 600 dpi Varityper (Figure 11-VT600), and the 1270 dpi Linotronic 300 (Figure 11-LT1270). The bitmapped font examples are formatted with TEX while the outline font examples are hand-coded POSTSCRIPT files sent directly to the printer. Since the formatter with which this paper is printed can use arbitrary POSTSCRIPT fonts, half of the examples could have been done in-line without pasting in. However, for fairness in the comparison, all examples were cut out and pasted in.

There are visible differences due to differences in the formatting software. TEX squeezes the letters closer together than does the POSTSCRIPT engine. Moreover, the interword space is constant in the POSTSCRIPT dictionary but is varied by TEX according to the line structure. These differences are not the differences that are at issue here.

On the 300 dpi device, the characters from the bitmapped fonts print thinner than are those of the outline fonts. However, the edges of both sets are equally smooth or jagged as the case may be in all sizes. Overall, then, the appearance of the characters of the bitmapped fonts is crisper than that of the outline fonts. On the higher resolution devices, the thicknesses of the characters are closer to being equal at all sizes. Thus, the METAFONT program does a better job of building a correctly sized bitmap at 300 dpi than does the 300 dpi POSTSCRIPT engine of the LaserWriterII. The latter seems to round up too much. However, both seem to get the edges equally smooth even at low sizes and low resolutions.

At the two higher resolutions, the outline fonts are significantly better than the outline fonts at lower resolutions and are significantly better than the bitmapped fonts at the same resolution of printing. However, this latter is true because the bitmapped fonts were generated by the METAFONT program specifically to be printed at 300 dpi. When a 300-dpi bitmap is printed with no scaling at 600 or 1270 dpi, it remains a 300-dpi bitmap. As expected, the 300-dpi bitmapped fonts print better at 300 dpi than they do at the two higher resolutions.

The generated outlines are not fine-tuned for printing at low resolutions, such as 300 dpi, as are the METAFONT-generated bitmaps. It might be useful to make use of the POSTSCRIPT facilities for hinting to improve the appearance of the characters printed from the outlines at low resolutions.

Figure 12 shows samples of similar sentences printed on the same three devices using the standard Helvetica, Times Roman, and Courier POSTSCRIPT outline fonts built into most POSTSCRIPT-executing laser printers. It appears to these authors that the standard POSTSCRIPT fonts are significantly better than those generated from METAFONT fonts. However, this is not surprising. Adobe uses a grid of 1000×1000 for its character definitions, resulting in a resolution of 7200 dpi for characters printed at point size 10. Because of size limitations of the METAFONT program the METAFONT outline fonts are using a resolution of 3,000 points per inch. However, when using the letters in small sizes such as from 10 to 70, quality differences are hardly visible especially when working with printers that have a resolution of 300 points per inch such as the Apple LaserWriter. Moreover, Adobe makes liberal use of hinting to improve the appearance of its fonts at low resolutions. We completely ignored hinting, as we did not see any way to automatically generate the hints.

6.2 Sizes of fonts. Recall that it is necessary to compare the size of the POSTSCRIPT outline font for a particular METAFONT definition to the sizes of the bitmapped fonts in POSTSCRIPT fonts for the individual and all magnifications.

This comparison is made in this section for the `cmr10` font at the standard set of six magnifications 1, 1.095, 1.2, 1.44, 1.728, and 2.07 (which are approximations of 1.2 raised to the powers 0, .5, 1, 2, 3, and 4, respectively). In addition, as a gesture to those who are not purists and accept magnifications of the 10 point design size as different point sizes, the comparison includes the `cmr` font at point size 5, 6, 7, 8, 9, 10, 12, and 17, the standard eight design sizes maintained for use with TEX.

R	M	7:	THIS IS CMR SEVEN POINTS WRITTEN IN METAFONT
		10:	THIS IS CMR TEN POINTS WRITTEN IN METAFONT
		12:	THIS IS CMR TWELVE POINTS WRITTEN IN METAFONT
	P	7:	THIS IS CMR SEVEN POINTS WRITTEN IN POSTSCRIPT
		10:	THIS IS CMR TEN POINTS WRITTEN IN POSTSCRIPT
		12:	THIS IS CMR TWELVE POINTS WRITTEN IN POSTSCRIPT
T	M	8:	THIS IS CMTT EIGHT POINTS WRITTEN IN METAFONT
		10:	THIS IS CMTT TEN POINTS WRITTEN IN METAFONT
		12:	THIS IS CMTT TWELVE POINTS WRITTEN IN METAFONT
	P	8:	THIS IS CMTT EIGHT POINTS WRITTEN IN POSTSCRIPT
		10:	THIS IS CMTT TEN POINTS WRITTEN IN POSTSCRIPT
		12:	THIS IS CMTT TWELVE POINTS WRITTEN IN POSTSCRIPT
L	M	10:	$\sqsubset\sqsupset\ \Diamond\Box\ \Join\leadsto\ \lhd\ \unlhd\ \rhd\ \unrhd$
	P	10:	$\sqsubset\sqsupset\ \Diamond\Box\ \Join\leadsto\ \lhd\ \unlhd\ \rhd\ \unrhd$

Figure 11-LW300

R	M	10:	THIS IS CMR TEN POINTS WRITTEN IN METAFONT
		12:	THIS IS CMR TWELVE POINTS WRITTEN IN METAFONT
	P	10:	THIS IS CMR TEN POINTS WRITTEN IN POSTSCRIPT
		12:	THIS IS CMR TWELVE POINTS WRITTEN IN POSTSCRIPT
T	M	10:	THIS IS CMTT TEN POINTS WRITTEN IN METAFONT
		12:	THIS IS CMTT TWELVE POINTS WRITTEN IN METAFONT
	P	10:	THIS IS CMTT TEN POINTS WRITTEN IN POSTSCRIPT
		12:	THIS IS CMTT TWELVE POINTS WRITTEN IN POSTSCRIPT
L	M	10:	$\sqsubset\sqsupset\ \Diamond\Box\ \Join\leadsto\ \lhd\ \unlhd\ \rhd\ \unrhd$
	P	10:	$\sqsubset\sqsupset\ \Diamond\Box\ \Join\leadsto\ \lhd\ \unlhd\ \rhd\ \unrhd$

Figure 11-VT600

R M 10: THIS IS CMR TEN POINTS WRITTEN IN METAFONT
12: THIS IS CMR TWELVE POINTS WRITTEN IN METAFONT

P 10: THIS IS CMR TEN POINTS WRITTEN IN POSTSCRIPT
12: THIS IS CMR TWELVE POINTS WRITTEN IN POSTSCRIPT

T M 10: `THIS IS CMTT TEN POINTS WRITTEN IN METAFONT`
12: `THIS IS CMTT TWELVE POINTS WRITTEN IN METAFONT`

P 10: `THIS IS CMTT TEN POINTS WRITTEN IN POSTSCRIPT`
12: `THIS IS CMTT TWELVE POINTS WRITTEN IN POSTSCRIPT`

L M 10: ⊏⊐ ◇□ ⋈⇝ ◁ ⊴ ▷ ⊵

P 10: ⊏⊐ ◇□ ⋈⇝ ◁ ⊴ ▷ ⊵

Figure 11-LT1270

Editor's note: See page 537 for discussion of resolution in METAFONT samples.

LW300: THIS IS HELVETICA TEN POINTS WRITTEN IN POSTSCRIPT
THIS IS HELVETICA TWELVE POINTS WRITTEN IN POSTSCRIPT

THIS IS TIMES-ROMAN TEN POINTS WRITTEN IN POSTSCRIPT
THIS IS TIMES-ROMAN TWELVE POINTS WRITTEN IN POSTSCRIPT

`THIS IS COURIER TEN POINTS WRITTEN IN POSTSCRIPT`
`THIS IS COURIER TWELVE POINTS WRITTEN IN POSTSCRIPT`

VT600: THIS IS HELVETICA TEN POINTS WRITTEN IN POSTSCRIPT
THIS IS HELVETICA TWELVE POINTS WRITTEN IN POSTSCRIPT

THIS IS TIMES-ROMAN TEN POINTS WRITTEN IN POSTSCRIPT
THIS IS TIMES-ROMAN TWELVE POINTS WRITTEN IN POSTSCRIPT

`THIS IS COURIER TEN POINTS WRITTEN IN POSTSCRIPT`
`THIS IS COURIER TWELVE POINTS WRITTEN IN POSTSCRIPT`

LT1270: THIS IS HELVETICA TEN POINTS WRITTEN IN POSTSCRIPT
THIS IS HELVETICA TWELVE POINTS WRITTEN IN POSTSCRIPT

THIS IS TIMES-ROMAN TEN POINTS WRITTEN IN POSTSCRIPT
THIS IS TIMES-ROMAN TWELVE POINTS WRITTEN IN POSTSCRIPT

`THIS IS COURIER TEN POINTS WRITTEN IN POSTSCRIPT`
`THIS IS COURIER TWELVE POINTS WRITTEN IN POSTSCRIPT`

Figure 12

Table 1 shows the sizes in bytes. Thus it is clear that the POSTSCRIPT outline font is bigger than any bitmapped font and that goal 4 fails. Moreover, it is clear that the outline font is bigger than the sum over all magnifications of one design size and than the sum over all standard design sizes. Thus goal 5 fails. In fact, this failure is the reason that the samples of Figure 11 involve only upper case letters. Samples with complete fonts with both cases often overloaded the printer available to the students at the time this work was done.

Font	Design size	Magnification	Bitmap (size in bytes)	Outlines (size in bytes)
cmr	10	1.0	22,812	245,000
"	10	1.095	24,231	"
"	10	1.2	26,044	"
"	10	1.44	31,892	"
"	10	1.728	39,614	"
"	10	2.07	50,578	"
cmr	5	1.0	16,729	"
"	6	1.0	17,757	"
"	7	1.0	18,820	"
"	8	1.0	20,041	"
"	9	1.0	21,580	"
"	12	1.0	25,658	"
"	17	1.0	37,140	"
Total			352,896	245,000

Table 1

However, do note that the outline font is smaller than the sum over all design sizes and magnifications thereof.

So in terms of disk space for the non-purists, the outline font represents a savings. Again notice that not all magnifications of the bitmapped fonts are maintained and the outline font is arbitrarily scaleable. Moreover, as the magnification grows the size of the bitmap grows even more rapidly.

The disappointment with respect to saving printer and disk memory says that it is important to spend more effort to optimize the outline font.

All is not lost, though! As this paper was being prepared for publication in *TUGboat*, one reviewer, Nelson Beebe, pointed out something that we can only kick ourselves for not noticing. The POSTSCRIPT outline fonts that are generated by mf2ps are horrendously wasteful in space. They use original, built-in command names and absolute coordinates. A significant reduction in size can be obtained by definition and use in the outlines of single-character command names, e.g., "M" for "moveto", and by use of relative versions of these commands with operands of fewer digits after the initial absolute moveto of any character. A simple filter was written to obtain new compressed versions of the POSTSCRIPT outline fonts. The appearances of the output when printing with these new versions is unchanged, but what is sent to the printer is significantly smaller, about 37.7% smaller. The reduction on a per-letter basis is about 45%. Table 2 shows the information of Table 1 for the new versions of the outline fonts.

Font	Design size	Magnification	Bitmap (size in bytes)	Outlines (size in bytes)
cmr	10	1.0	22,812	152,670
"	10	1.095	24,231	"
"	10	1.2	26,044	"
"	10	1.44	31,892	"
"	10	1.728	39,614	"
"	10	2.07	50,578	"
cmr	5	1.0	16,729	"
"	6	1.0	17,757	"
"	7	1.0	18,820	"
"	8	1.0	20,041	"
"	9	1.0	21,580	"
"	12	1.0	25,658	"
"	17	1.0	37,140	"
Total			352,896	152,670

Table 2

There are still better compressions that can be achieved. According to Beebe [4], Toal and Raine's outline representation of cmr at 10 points requires about twice the space needed for bitmaps of the same; at 14 to 16 points, the outlines and the bitmaps occupy about the same amount of space; above 16 points, the outlines are smaller than the bitmaps. It is clear that better encodings exist than we explored and these must be explored for any future version of mf2ps.

One such better encoding appears to be that used by Adobe for its own proprietary fonts; fonts encoded this way have a FontType of 1. User defined fonts have a FontType of 3. Beebe [4] says that type 1 fonts are handled with greater efficiency than type 3 fonts on most existing POSTSCRIPT interpreters, especially those that are based on Adobe-licensed code. Adobe has recently published specifications for the type 1 font encoding [2], thus allowing anyone to produced type 1 fonts. Beebe believes that the market forces will drive other companies to encode their fonts as type 1. Moreover, as more and more windowing systems based on POSTSCRIPT, e.g., NeWS and NeXT, appear, the attraction of POSTSCRIPT outline fonts will increase, as then the same font can be used for both printing and previewing. Thus, the incentive will be to convert META-

FONT fonts into type 1 POSTSCRIPT outline fonts.

Ultimately, the tradeoff is between the size of the font sent to the printer, and the time it takes for the printer to decode the program for the characters. However, with proper cacheing, a big enough cache, and a not very fancy document, the decoding is done only once per character for the document!

7 Future work

For the future, there are a number of improvements that can be made. Currently, each letter of the POSTSCRIPT outline fonts is described as a set of cyclic paths. When all are filled or stroked, one gets the desired letter. Some of those cyclic paths have a common boundary that is inside the letter and is not necessary for the outline description of the letter as a whole. Eliminating these paths and creating one outline for the letter will save space. Today this can be done manually, and is worth the effort because the translation process is done only once. From that time on, the font is used the way it is.

As was demonstrated by Beebe's rescue of our result, closer attention should be paid to obtaining more compact representations of character outlines, representations for which POSTSCRIPT routines can be written to interpret them into standard outline drawing commands. Collapsing commands into single characters and using relative movements saved significant amounts of space. Perhaps, even more dramatic savings can be obtained by giving coordinates and distances in hexadecimal.

More effort can be spent on modifying the program in order to allow magnifications up to 7200 points. Thus, no jaggies will be seen, as occasionally happens when using higher magnifications, e.g., in our translated fonts at magnification 8. This could be done by enlarging the program arrays to handle characters based on 7200 points. A sophisticated solution is required if one wants to save room while compiling the input font. In such a case, any linear translation which is done within the POSTSCRIPT program is with a factor less than 1.

METAFONT was changed for TEX 3.0. It is necessary to build a new version of mf2ps based on this latest version of METAFONT. As the changes to the METAFONT program deal mainly with ligatures and kerning, the calculation of envelopes is probably not affected. Therefore, it is likely that the portion of METAFONT up to the calculation of the envelope can still be used as a front end for mf2ps with very little change in the portion of the program we wrote.

Finally, it might be worthwhile, for the sake of portability to other systems and enhanceability by other humans, to rewrite or to write the next version of mf2ps with WEB.

Acknowledgments

The authors thank the *TUGboat* editors, and Nelson Beebe for their help, sharp comments, and result-saving ideas. Dealing with their comments made this a better paper.

References

1. *POSTSCRIPT Language Reference Manual,* Adobe Systems Incorporated, Addison-Wesley, Reading, MA (1985).
2. "Adobe Type 1 Font Format," Part No. LPS0064, Adobe Systems, Inc. (March, 1990).
3. S. von Bechtolsheim, "The TEX PostScript Software Package," *TUGboat* **10**(1), p. 25–27 (1989).
4. N. Beebe, Private communication, via electronic mail. (1990).
5. L. Carr, "Of Metafont and PostScript," *TEXniques* **5**, p. 141–152 (August, 1987).
6. D. Henderson, "Outline fonts with METAFONT," *TUGboat* **10**(1), p. 36–38 (1989).
7. J.D. Hobby, "A METAFONT-like System with PostScript Output," *TUGboat* **10**(4), p. 505–512 (1989).
8. B.W. Kernighan, "A Typesetter-independent TROFF," Computing Science Technical Report No. 97, Bell Laboratories, Murray Hill, NJ 07974 (March, 1982).
9. D.E. Knuth, *The TEXbook,* Addison-Wesley, Reading, MA (1984).
10. D.E. Knuth, *The* METAFONT*book,* Addison-Wesley, Reading, MA (1986).
11. D.E. Knuth, METAFONT*: The Program,* Addison-Wesley, Reading, MA (1987).
12. G. Toal, Private communication, via electronic mail. (1990).
13. J. Ziv and A. Lempel, "A Universal Algorithm for Sequential Data Compression," *IEEE Transactions on Information Theory* **3**, p. 337–343 (May, 1977).

◊ Shimon Yanai
IBM Science and Technology Center
Technion City
Haifa 32000
Israel
yanai@israearn.bitnet

◊ Daniel M. Berry
Computer Science
Technion
Haifa 32000
Israel
dberry@cs.technion.ac.il

An Improved Chess Font

David Tofsted

In the July 1989 issue of *TUGboat* an article by Zalman Rubinstein discussed how to display chess board positions using a font created with METAFONT. Upon reading this article I was interested enough in the idea of producing a high quality chess font that I felt I had to try my hand at improving on Professor Rubinstein's "first cut." The model I used for the set was derived from a combination of the chess characters used by the *New York Times* and those of a vintage chess tutorial I have at home. The results of this work are presented here along with some comments concerning METAFONT peculiarities that were overcome in developing the characters. I do hope there is enough interest in the TEX community to use this new font. Perhaps it will also inspire others as to the highly versatile nature of the METAFONT program.

To begin, consider the net results. Figure 1 shows the 26 possible characters in this font. These include the six different pieces of each side on two differently colored squares each, plus an empty black and an empty white square. The commands used to present the chess board can be obtained from either Prof. Rubinstein's article or the article by Mr. Wolfgang Appelt (December 1988, *TUGboat*).

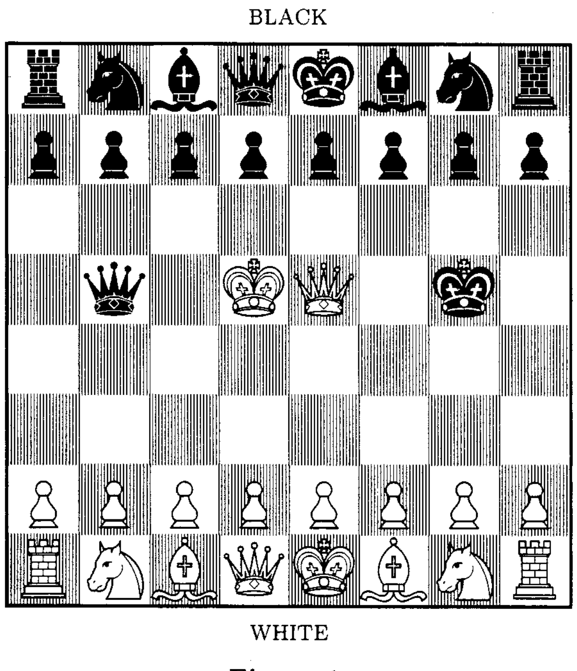


Figure 1.

In describing how these characters are formed, it must first be noted that the original printing format was a LaserJet, Series II. At 300 dpi the characters are amazingly sharp, but the hatched background (lines 28-33)* had to be handled carefully. It was designed such that every fifth column of dots was turned on. This limited the METAFONTness of the resulting font because changes were required at each magnification. Having a column of filled-in dots every five columns required adjusting the looping factor on the `hatch` subroutine. Without this adjustment the METAFONT program performs roundoff that causes the line spacing to be uneven. Also, the LaserJet tends to overlap the space between dots. This overfilling meant that the erase draws performed in the interior of the black characters had to be two dots wide rather than one, to compensate for the overfilling.

Aside from this minor detail, the characters are drawn similarly for both black and white pieces. The same set of control points can be used for each (the pair function `w[]`). Thus the array assignments only need to be specified once (lines 46-59) for each piece type. These points are then converted into `z` points (lines 64, 99, 119, and 143) within the character description. Additionally, the total number of characters is 26, but the number of chess pieces that must actually be drawn is only 12. This is possible because of Donald Knuth's `keepit` "dirty trick" (*The METAFONTbook*, p. 295) (lines 4-14, 88, and 98 for example). Using Knuth's technique, the chess piece is first drawn on a white background and saved (88, 98), then the next character (same piece on a black background) can be drawn simply by calling the `hatch` subroutine into a fresh picture drawing area (105), erasing out the area of the hatched character that will be filled in by the previously drawn piece (106-112), and then using the `addto` command (113) to combine the original chess piece with the new background.

Space would not allow a full listing and description of all the characters, so I will discuss in detail the steps required to generate only the Bishop, leaving the rest as an exercise for the reader. Before proceeding though, it would seem appropriate to comment on some of the key aspects of each of the other pieces as well. The two simplest characters were the Rook and the Queen. Both are symmetric about the vertical center line. Both were composed almost entirely of straight line segments. The Rook was particularly simple in that it used several points along the vertical center line and then achieved the brick corners via `penpos` commands.

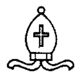
*This notation refers to line numbers in the listing at the end of this article. Other references to the listing will be designated similarly.

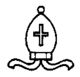
A third piece, the King, was only difficult because of the number of points involved. Over 50 were needed. One problem area that did arise was in producing the curved lines describing the King's upper crown. Control points had been chosen in the arc around the two upper lobes of the crown, but when drawing the crown the tension command was needed on either side of the center to keep the lines from overlapping as in the example below. (A tension of 1.2 was found sufficient to correct the problem.)

Of the remaining three pieces, producing the Knight was rather simple because of its asymmetric shape. The asymmetry meant there was a wider range of shapes that "looked" right. I must however credit my wife Laura for the original drawing of the horse used in the Knight. The Pawn I finally adopted is possibly too simple, but this was a design decision since it focuses more attention on the stronger pieces.

This leaves the Bishop. Oddly enough more time was spent getting this character right than any of the others. At first a doubled miter was tried, but this construct resulted in a lopsided look. Also I had originally defined twelve points to use as turning points in drawing the cross in the miter. But this approach led to problems on the 300 dpi machine. I cannot reproduce this problem for this article because of the higher printer resolution, but suffice it to say that the vertical lines in the upper extension of the cross did not match the line in the lower cross portion. The vertical section therefore appeared crooked in the middle. This problem was remedied by using `fill` and `unfill` commands (lines 82-85) instead of a single draw command.

A final problem was discovered after the other characters had been produced. This problem is illustrated by the Bishop below on the left.





The Bishop on the left has a small overlap of the lines at the miter's base. The solution was to use the up directional command at that point (75 and 78).

Given this description, the complete code used to compose all four versions of Bishop is included below.

```
 1. %Components needed for D. Knuth's
 2. %KEEPIT dirty trick.
 3. %
 4. picture extrapic;
 5. boolean currentnull, extranull;
 6. def clearit = currentpicture:= extrapic;
 7.  currentnull:=extranull;
 8.  extrapic:=nullpicture;
 9.  extranull:=true; enddef;
10.
11. def keepit =
12.  cull currentpicture keeping (1,infinity);
13.  extrapic:=currentpicture;
14.  extranull:=currentnull; enddef;
15.
16. %Define LaserJet device parameters.
17. %
18. mode_setup;
19.  em#:=1/3in#;
20.  thin#:=.01em#; thick#:=.02em#;
21.  define_pixels(em,cap,ext,dep);
22.  define_blacker_pixels(thin,thick);
23.  curve_sidebar=round 1/18em;
24.
25. %Draw vertical hatching background for
26. %pieces on black squares.
27. %
28. def hatch(expr dummy) =
29.  pickup pencircle scaled thin;
30.  for i=0 upto 22:
31.   draw (w*i/22,0)--(w*i/22,h);
32.  endfor
33.  enddef;
34.
35. %Point locations as a percentage of the
36. %full character width.  #'s 1-5 define
37. %the left cloth strip, 6-10 the right
38. %cloth strip, 11-16 the left side of
39. %the miter, 16-17 the top circle, 18-20
40. %and 26-27 the right side of the miter,
41. %21-25 were are an unused second miter
42. %section, 28-29 details at the base of
43. %the miter, 30-37 cross in the center
44. %of the miter.
45. %
46. pair w[];
47.  w1=(7,9); w2=(10,15);
48.  w3=(16,16); w4=(38,10);
49.  w5=(46.5,20); w6=(53.5,20); w7=(62,10);
50.  w8=(84,16);   w9=(90,15);   w10=(93,9);
51.  w11=(35,20);  w12=(35,30);  w13=(29,43);
52.  w14=(29,55);  w15=(38,75);  w16=(50,86);
53.  w17=(50,95);  w18=(62,75);  w19=(71,55);
54.  w20=(71,43);  w21=(65,75);  w22=(61,73);
55.  w23=(72,55);  w24=(69,55);  w25=(69,43);
56.  w26=(65,30);  w27=(65,20);  w28=(42,30);
57.  w29=(58,30);  w30=(39,56);  w31=(61,56);
58.  w32=(50,39);  w33=(50,70);  w34=(41,56);
59.  w35=(59,56);  w36=(50,41);  w37=(50,68);
```

```

%The White Bishop on a White Square
%
beginchar(4,em#,em#,0); "White Bishop";
 for i=1 upto 37: z[i]=h/100*w[i]; endfor
 penpos1(4thick,-20); penpos2(4thick,-50);
 penpos3(4thick,-70); penpos4(4thick,-70);
 penpos5(4thick,0);   penpos6(4thick,0);
 penpos7(4thick,60);  penpos8(4thick,60);
 penpos9(4thick,50);  penpos10(4thick,40);
 penpos30(3thick,90); penpos31(3thick,90);
 penpos32(3thick,0);  penpos33(3thick,0);
 penpos34(thick,90);  penpos35(thick,90);
 penpos36(thick,0);   penpos37(thick,0);
 pickup pencircle scaled thick;
 draw z1r..z2r..z3r..z4r..{up}.5[z5,z6]--
      z5l..z4l..z3l..z2l..z1l--cycle;
 draw z6r..z7r..z8r..z9r..z10r--z10l..z9l..
      z8l..z7l..{up}.5[z5,z6]--cycle;
 draw z11--z12..z13..z14..z15..z16--z16..
      z18..z19..z20..z26--z27--cycle;
 draw z16..z17..cycle;
 fill z30r--z31r--z31l--z30l--cycle;
 fill z32r--z33r--z33l--z32l--cycle; cullit;
 unfill z34r--z35r--z35l--z34l--cycle;
 unfill z36r--z37r--z37l--z36l--cycle;
 pickup pencircle scaled thin;
 draw z12--z28--z5l--z28--z29--z6r--z29--z26;
showit; keepit;
endchar;

%The White Bishop on a Black Square
%(All that is necessary is hatching,
%unfilling the area to be filled be the
%previously drawn character, and using
%addto.)
%
beginchar(10,em#,em#,0); "Wht Bish on Blk";
 keepit; currentpicture:=nullpicture;
 for i=1 upto 37: z[i]=h/100*w[i]; endfor
 penpos1(4thick,-20); penpos2(4thick,-50);
 penpos3(4thick,-70); penpos4(4thick,-70);
 penpos5(4thick,0);   penpos6(4thick,0);
 penpos7(4thick,60);  penpos8(4thick,60);
 penpos9(4thick,50);  penpos10(4thick,40);
 hatch(1);
 unfill z1r..z2r..z3r..z4r..z5r--z5l..z4l..
        z3l..z2l..z1l--cycle;
 unfill z6r..z7r..z8r..z9r..z10r--z10l..
        z9l..z8l..z7l..z6l--cycle;
 unfill z11--z12..z13..z14..z15..z16--z16..
        z18..z19..z20..z26--z27--cycle;
 unfill z16..z17..cycle; cullit;
 addto currentpicture also extrapic;
 pickup pencircle scaled thick;
showit;
endchar;

beginchar(16,em#,em#,0); "Black Bishop";
 for i=1 upto 33: z[i]=h/100*w[i]; endfor
 penpos1(4thick,-20); penpos2(4thick,-50);
 penpos3(4thick,-70); penpos4(4thick,-70);
 penpos5(4thick,0);   penpos6(4thick,0);
 penpos7(4thick,60);  penpos8(4thick,60);
 penpos9(4thick,50);  penpos10(4thick,40);
 penpos30(2thick,90); penpos31(2thick,90);
 penpos32(2thick,0);  penpos33(2thick,0);
 fill z1r..z2r..z3r..z4r..z5r--z5l..z4l..
      z3l..z2l..z1l--cycle;
 fill z6r..z7r..z8r..z9r..z10r--z10l..
      z9l..z8l..z7l..z6l--cycle;
 fill z11--z12..z13..z14..z15..z16--z16..
      z18..z19..z20..z26..z27--cycle;
 fill z16..z17..cycle;
 cullit;
 unfill z30r--z31r--z31l--z30l--cycle;
 unfill z32r--z33r--z33l--z32l--cycle;
 pickup pencircle scaled thick;
 erase draw z12--z26--z29--z6r--z5l--z28;
showit; keepit;
endchar;

beginchar(22,em#,em#,0); "Blk Bish on Blk";
 for i=1 upto 33: z[i]=h/100*w[i]; endfor
 penpos5(4thick,0); penpos6(4thick,0);
 penpos30(2thick,90); penpos31(2thick,90);
 penpos32(2thick,0);  penpos33(2thick,0);
 hatch(1);
 unfill z30r--z31r--z31l--z30l--cycle;
 unfill z32r--z33r--z33l--z32l--cycle;
 pickup pencircle scaled thick;
 erase draw z12--z26--z29--z6r--z5l--z28;
showit;
endchar;

stop"";
end;
```

To conclude, the chess font described was designed to be useful for a variety of chess publication needs. It is hoped the font provided is robust enough to avoid others having rework this same problem later. I also hope this example of the METAFONT program's capabilities should stimulate interest in other applications along these lines. One such application might extend TEX into the area of archeology by way of a font for hieroglyphics or cuneiform.

Any comments, suggestions, or requests regarding this font are welcome. I can be reached at the address below. Unfortunately I do not have access to electronic mail.

◇ David Tofsted
P. O. Box 6926
Las Cruces, NM 88006

Output Devices

TeX Output Devices

Don Hosek

Introduction

The number of device drivers (especially in the UNIX world) and proliferation of distribution venues for those drivers has caused it to be impossible to retain the old format for the driver listings and provide a useful amount of information (not to mention the difficulties in maintaining such a monster). The listings are in the process of being installed into a database to simplify answering driver queries and maintenance of information; this should allow future occurrences of these listings to be somewhat timelier.

The information is now broken down into four sections, one for each of laser xerographic printers, impact printers, phototypesetters, and screen displays. The listings are first by output device then by computer hardware, except for the previewers which are listed by computer. In those cases where a driver for a given printer runs on more than one computer, the description of the driver is listed just under the name of the printer and cross-reference is made to it under each computer on which it runs. Difficult-to-classify drivers (*e.g.*, those which, rather than drive printers directly, drive some generic graphic interface) are put at the end of the impact printer section for lack of a better place. All suppliers are given in a final section.

The old tables have been replaced by simplified tables which indicate the existence of a driver for a given printer/computer combination by referring to the page number on which that combination's listing begins. If no page number is present, we are unaware of a driver for the combination. These tables, which begin on page 567, can therefore be used as an index into the listings.

In coming volumes of *TUGboat*, the complete driver listings will appear in the first regular issue with updates published as necessary in subsequent regular issues for that year.

As before, corrections, updates, and new information for the list are welcome; they may be sent to me at `dhosek@ymir.claremont.edu` or via postal mail to the address listed on 483.

Contents

Drivers for Laser Xerographic and Electro-Erosion Printers

Agfa P400

DVIP400 (by Bernd Schulze). Uses PXL files. Allows landscape printing and inclusion of P400 bitmap graphics. Written in WEB. Source available on request.

• IBM MVS

DVIP400 (by Bernd Schulze). See description above. Cost: 300–1848DM. Suppliers: Systemhaus für Elektronisches Publizieren.

• IBM PC

DVIP400 (by Bernd Schulze). See description above. Cost: 300–1848DM. Suppliers: Systemhaus für Elektronisches Publizieren.

• IBM VM/CMS

DVIP400 (by Bernd Schulze). See description above. Cost: 300–1848DM. Suppliers: Systemhaus für Elektronisches Publizieren.

- Siemens BS2000

Unspecified program. Suppliers: Universität des Saarlandes.

- Unix

DVIP400 (by Bernd Schulze). See description above. Cost: 300–1848DM. Suppliers: Systemhaus für Elektronisches Publizieren.

Unspecified program. Suppliers: Universität des Saarlandes.

- VAX/VMS

DVIP400 (by Bernd Schulze). See description above. Cost: 300–1848DM. Suppliers: Systemhaus für Elektronisches Publizieren.

Canon LBP-A2, LBP-8

DVICAN (by Nelson H. F. Beebe). Uses GF, PK or PXL files. Written in C. Source is included.

- Atari ST

DVICAN (by Nelson H. F. Beebe). See description above. Suppliers: FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩, University of Utah.

- DEC-20

DVICAN (by Nelson H. F. Beebe). See description above. Suppliers: FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩.

- IBM PC

DVICAN (by Nelson H. F. Beebe). See description above. Suppliers: FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩, Radel, Personal TEX.

- Unix

DVICAN (by Nelson H. F. Beebe). See description above. Suppliers: FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩, University of Utah.

Unspecified program. Suppliers: Canon.

- VAX/VMS

DVICAN (by Nelson H. F. Beebe). See description above. Suppliers: FTP ⟨ctrsci.utah.edu⟩, University of Utah.

Cordata LP300

- IBM PC

PC Laser/Cordata. Requires 512K RAM disk. Cost: $195. Suppliers: Personal TEX.

DEC LN03, LN03+

DVIL3P (by John Sauter). Uses GF, PK, and PXL files. Written in C. Source is included.

- Atari ST

DVIL3P (by John Sauter). See description above. Suppliers: FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩, University of Utah.

- DEC-20

DVIL3P (by John Sauter). See description above. Suppliers: FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩.

- IBM PC

DVI2LN3 (by Flavio Rose, modified by Stanley Sawyer). Uses all three PC PXL formats and PK files. Graphics specials for line drawing included. The driver will scale fonts by multiples of 2 or 3 if no closer size would be available. Written in C. Source included. The program is distributed free of charge with the receipt of a blank disk and return mailer. Suppliers: Washington University.

DVIL3P (by John Sauter). See description above. Suppliers: FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩, Radel, Personal TEX.

- Unix

DVIL3P (by John Sauter). See description above. Suppliers: FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩, University of Utah.

- VAX/VMS

DVI2LN3 (by Flavio Rose; modified by Edwin Bell). Uses PXL files. Allows inclusion of Sixel graphics in two formats. Written in C. Distributed in source format. Suppliers: University of Kansas.

DVIL3P (by John Sauter). See description above. Suppliers: FTP ⟨ctrsci.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩, University of Utah.

DVItoLN03 3.0 (by Brian Hamilton Kelly). Uses PK and PXL files. Written in WEB. Source is included. The program is accessed through the standard DCL interface. Font downloading is on a per character basis rather than a per font basis to conserve printer memory. Large characters are printed as downloaded graphics. Support for invisible fonts and 256 character fonts is provided. The driver does not require additional RAM cartridges for the printer, but it helps. Suppliers: Aston, DECUS TEX collection, FTP ⟨uk.ac.aston.tex⟩, FTP ⟨ymir.claremont.edu⟩.

T2/LN03. Uses GF or PK files. Distributed as executable (VMS 4.6 or later). Supports use of LN03 internal fonts and inclusion of LN03 graphics and illustrations. A RAM cartridge is suggested for optimal performance. Cost: $495 (1600bpi magtape), $515 (TK50 cartridge). Suppliers: Northlake Software.

Unspecified program. Suppliers: Procyon Informatics.

Golden Dawn Golden Laser 100

DVIGD (by Nelson H. F. Beebe). Uses GF, PK, and PXL files. Written in C. Source is included.

• Atari ST

DVIGD (by Nelson H. F. Beebe). See description above. Suppliers: FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩, University of Utah.

• DEC-20

DVIGD (by Nelson H. F. Beebe). See description above. Suppliers: FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩.

• IBM PC

DVIGD (by Nelson H. F. Beebe). See description above. Suppliers: FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩, Radel, Personal TEX.

• Unix

DVIGD (by Nelson H. F. Beebe). See description above. Suppliers: FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩, University of Utah.

• VAX/VMS

DVIGD (by Nelson H. F. Beebe). See description above. Suppliers: FTP ⟨ctrsci.utah.edu⟩, University of Utah.

HP 2680

• HP 1000

Unspecified program. Suppliers: JDJ Wordware.

HP 2688A

• HP 1000

Unspecified program. Suppliers: JDJ Wordware.

• HP 9000/200

Unspecified program. Suppliers: Hewlett-Packard.

HP LaserJet, LaserJet Plus, II, IID, IIP, III, 2000

DVI2XX (by Gustav Neumann). Uses PK or PXL files. Written in C. Source is included. Supports odd and even-only page printing (for two-run duplex). Graphics inclusion is also supported.

DVIJE2. Uses GF, PK or PXL files. Written in C. A modified version of DVIJEP by Nelson H. F. Beebe optimized for the LaserJet Series II.

DVIJEP (by Nelson H. F. Beebe). Uses GF, PK or PXL files. Written in C. Source is included. Graphics inclusion specials are available on request.

DVIlaser/HP. Uses GF, PK, or PXL files. Allows inclusion of graphics, use of printer resident fonts, font substitution, font scaling, and magnifies or shrinks images.

• Amiga

Unspecified program. Uses PK files. Allows landscape printing. Suppliers: Radical Eye Software.

• Atari ST

DVIJE2. See description above. Suppliers: FTP ⟨ymir.claremont.edu⟩.

DVIJEP (by Nelson H. F. Beebe). See description above. Suppliers: FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩, University of Utah.

Unspecified program. Cost: £100. Suppliers: Oxford [2].

Unspecified program. Suppliers: TEXsys.

Unspecified program. Suppliers: Tools GmbH Bonn.

• DEC-20

DVIJE2. See description above. Suppliers: FTP ⟨ymir.claremont.edu⟩.

DVIJEP (by Nelson H. F. Beebe). See description above. Suppliers: FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩.

• HP 1000

Unspecified program. Suppliers: Technical Research Center of Finland.

• IBM PC

DVI2XX (by Gustav Neumann). See description above. Suppliers: Neumann, FTP ⟨uk.aston.ac.uk⟩.

DVIHPLJ (by Eberhard Mattes). Uses PK or PXL files. Supports VF files and graphics inclusion. Suppliers: Aston, FTP ⟨rusmv1.rus.uni-stuttgart.de⟩, FTP ⟨terminator.cc.umich.edu⟩, FTP ⟨uk.ac.aston.tex⟩, Radel.

DVIJE2. See description above. Suppliers: FTP ⟨ymir.claremont.edu⟩.

DVIJEP (by Nelson H. F. Beebe). See description above. Suppliers: FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩, Radel, Personal TEX.

DVIlaser/HP. See description above. Allows use of HP soft fonts. Cost: $225. Site licenses available. Academic discounts available. Suppliers: ArborText, Inc., TEX Users Group.

PTI Laser/HP. Allows automatic font substitution, landscape printing, and inclusion of graphics. Cost: $195. Suppliers: Personal TEX.

Unspecified program. Suppliers: LaserPrint.

Unspecified program. Suppliers: XOrbit.

• Prime

DVI2LJ (by Tor Lillquist; ported to Primos by Marc-Rene Uchida). Uses PXL files. Written in Pascal and PLP. Source included. Suppliers: Prime distribution tape.

• Unix

dvi2lj (by Riccardo Mazza). Written in C. Source included. The program has been ported to VAX BSD 4.3, SCO i386 Unix and various 680x0 System V systems. It can only print on A4 paper. Suppliers: FTP ⟨orc.olivetti.com⟩.

DVI2XX (by Gustav Neumann). See description above. Runs on an HP9000/500. Suppliers: Neumann.

DVIJE2. See description above. Suppliers: FTP ⟨ymir.claremont.edu⟩.

DVIJEP (by Nelson H. F. Beebe). See description above. Suppliers: FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩, University of Utah.

DVIlaser/HP. See description above. Available for DEC/Unix, Apollo and Sun. Cost: $500 workstations; $750 mainframes. Site licenses available. Academic discounts available. Suppliers: ArborText, Inc.

Unspecified program. Written for HP 9000/500. Suppliers: Max-Planck-Institut für Aeronomie.

Unspecified program. Available for Ultrix and SunOS. Other ports available on request. Cost: £100. Suppliers: Oxford [2].

Unspecified program. Suppliers: Texas A&M [2].

• VAX/VMS

DVIJE2. See description above. Suppliers: FTP ⟨ymir.claremont.edu⟩.

DVIJEP (by Nelson H. F. Beebe). See description above. Suppliers: FTP ⟨ctrsci.utah.edu⟩, University of Utah.

DVIlaser/HP. See description above. Cost: $500 workstations; $750 mainframes. Site licenses available. Academic discounts available. Suppliers: ArborText, Inc.

T2/jet. Uses GF or PK files. Distributed as executable (VMS 4.6 or later). Allows inclusion of PCL graphics, landscape printing, use of HP built-in and cartridge fonts as well as downloadable soft fonts. Supports duplex printing on LaserJet IID and LaserJet 2000. Cost: $395 (1600bpi magtape), $415 (TK50 cartridge). Suppliers: Northlake Software.

Unspecified program. Suppliers: LaserPrint.

Unspecified program. Cost: £100. Suppliers: Oxford [2].

IBM 38xx, 4250, Sherpa

DVI2LIST (by Bob Creasy and Peter Sih). Uses IBM fonts. Comes with utility program for creating IBM fonts from PXLs. Graphics inclusion is supported.

DVI2XX (by Gustav Neumann). Uses PK or PXL files. Written in C. Source is included. Supports odd and even-only page printing (for two-run duplex).

DVIIBM. Uses PXL files. Supports landscape printing.

• IBM MVS

DVI2LIST (by Bob Creasy and Peter Sih; modified by Joachim Lammarsch). See description above. Suppliers: University of Heidelberg.

DVIIBM. See description above. Suppliers: Gesellschaft für Mathematik und Datenverarbeitung [1].

• IBM PC

DVI2XX (by Gustav Neumann). See description above. Suppliers: Neumann.

• IBM VM/CMS

DVI2LIST (by Bob Creasy and Peter Sih). See description above. Suppliers: Washington State University.

DVIIBM. See description above. Suppliers: Gesellschaft für Mathematik und Datenverarbeitung [1].

• Unix

DVI2XX (by Gustav Neumann). See description above. Runs on an HP 9000/500. Suppliers: Neumann.

Imagen

DVIIMP (by Lon Willett). Uses GF, PK or PXL files. Written in C. Source is included.

DVIlaser/IMP. Uses GF, PK, or PXL files. Allows inclusion of graphics and use of resident fonts, font substitution, font scaling, and magnifies or shrinks images. VF support included.

• Amdahl MTS

DVIlaser/IMP. See description above. Supported on 'as is' basis. Cost: $750. Site licenses available. Academic discounts available. Suppliers: ArborText, Inc.

Unspecified program. Suppliers: University of British Columbia.

• Atari ST

DVIIMP (by Lon Willett). See description above. Suppliers: FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩, University of Utah.

• Data General MV

Unspecified program. Suppliers: Texas A&M [1].

• DEC-20

DVIIMP (by Lon Willett). See description above. Suppliers: FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩.

Unspecified program. Suppliers: Columbia University.

• IBM PC

DVIIMP (by Lon Willett). See description above. Suppliers: FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩, Radel, Personal TEX.

DVIlaser/IMP. See description above. Supported on 'as is' basis. Cost: $225. Site licenses available. Academic discounts available. Suppliers: ArborText, Inc., TEX Users Group.

• IBM VM/CMS

DVIlaser/IMP. See description above. Supported on 'as is' basis. Cost: $750. Site licenses available. Academic discounts available. Suppliers: ArborText, Inc.

WIMPRESS. Uses RST files. A program to convert PXL files to RST files is included. Written in Pascal. Source is included. Suppliers: Weizmann.

DVITOIMP. Uses RST files. A program to convert PXL files to RST files is included. Allows inclusion of graphics, landscape printing and 2-up printing.

• Symbolics Lisp

dvi-stream (by Chris Lindblad). Written in Zetalisp. Source is included. Uses the Generic Hardcopy Interface to drive the Imagen printer. Supports landscape printing and graphics inclusion. Suppliers: Massachusetts Institute of Technology.

• Unix

DVIIMP (by Lon Willett). See description above. Suppliers: FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩, University of Utah.

DVIlaser/IMP. See description above. Supported on 'as is' basis. Available for DEC/Unix, Apollo and Sun. Cost: $750 multi-user systems, $500 workstations. Site licenses available. Academic discounts available. Suppliers: ArborText, Inc.

iptex (by Chris Torek). Uses GF, PK, or PXL files. Written in C with a front end written in Unix sh. Source is included. Supports landscape printing and variable printer resolution. A program for creating a DVI file containing a subset of pages of the original DVI file is included. Suppliers: University of Maryland.

Unspecified program. Available for the Apollo. Suppliers: OCLC.

Unspecified program. Suppliers: Sun.

• VAX/VMS

DVIIMP (by Lon Willett). See description above. Suppliers: FTP ⟨ctrsci.utah.edu⟩, University of Utah.

DVIlaser/IMP. See description above. Supported on 'as is' basis. Suppliers: ArborText, Inc.

IMPRINT. Uses GF and PK files. Can use printer resident fonts and print in landscape orientation. Cost: $1200 on a 600′ magtape at 1600bpi. Suppliers: Northlake Software.

Kyocera F-10xx, F-20xx

• Atari ST

Unspecified program. Suppliers: TEXsys.

• IBM PC

DVIHPLJ (by Eberhard Mattes). Uses PK or PXL files. Supports VF files and graphics inclusion. Suppliers: Aston, FTP ⟨rusmv1.rus.uni-stuttgart.de⟩, FTP ⟨terminator.cc.umich.edu⟩, FTP ⟨uk.ac.aston.tex⟩, Radel.

Unspecified program. Suppliers: LaserPrint.

• Unix

Unspecified program. Written in C. Suppliers: Max-Planck-Institut für Aeronomie.

• VAX/VMS

Unspecified program. Suppliers: LaserPrint.

Unspecified program. Written in C. Suppliers: Max-Planck-Institut für Aeronomie.

Océ 6750

• VAX/VMS

Unspecified program. See *TUGboat* 10, no. 1, pp. 56–58. Suppliers: Océ-Nederland.

Olympia Elsa

• IBM VM/CMS

DVIELSA (by Dr. Georg Bayer). Uses PXL files at 300dpi. Suppliers: Technische Universität Braunschweig.

PostScript printers

DVIALW (by Nelson H. F. Beebe and Neal Holtz). Uses GF, PK or PXL files. Graphics inclusion is supported. Written in C. Source is included.

DVIlaser/PS. Uses GF, PK, and PXL files. Allows inclusion of graphics, use of printer resident fonts, font substitution, font scaling, landscape printing and magnifies or shrinks images. A program AFtoTF is included for generating TFM files from AFM files. VF support included.

DVIPS (by Tom Rokicki). Uses PK files. Allows landscape printing, inclusion of PostScript graphics and use of internal and downloadable PostScript fonts. VF support is included. Written in C. Source included.

dvitops (by James Clark). Uses PK files. Allows use of printer-resident and downloaded PostScript fonts. Allows inclusion of graphics files and inline PostScript as well as arbitrary linear transformations to regions of the DVI file. Output is designed so that each page depends only on itself and the preamble. There is no device-dependence in the PostScript code. Included are programs for converting AFM files to PL files (*aftopl*) and for converting Adobe fonts from IBM PC format to straight PostScript (*afbtops*). Written in C. Distributed as source.

- Amiga

Unspecified program. Uses PK files. Allows inclusion of graphics and landscape printing. Suppliers: Radical Eye Software.

- Atari ST

DVIALW (by Nelson H. F. Beebe and Neal Holtz). See description above. Suppliers: FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩, University of Utah.

- DEC-20

DVIALW (by Nelson H. F. Beebe and Neal Holtz). See description above. Suppliers: FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩.

- IBM PC

DVIALW (by Nelson H. F. Beebe and Neal Holtz). See description above. Suppliers: FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩, Radel, Personal TEX.

DVIlaser/PS. See description above. Cost: $225. Site licenses available. Academic discounts available. Suppliers: ArborText, Inc., TEX Users Group.

dvitops (by James Clark). See description above. Suppliers: Aston, Clark, FTP ⟨uk.ac.aston.tex⟩.

PTI Laser/PS. Allows landscape printing, use of PostScript fonts, and inclusion of PostScript graphics. Cost: $195. Suppliers: Personal TEX.

- IBM VM/CMS

DVIlaser/PS. See description above. Cost: $750. Site licenses available. Academic discounts available. Suppliers: ArborText, Inc.

- Prime

DVI2PS (by Tang Tang). Uses PXL files. Written in C. Source included. Suppliers: Prime distribution tape.

DVIALW (by Mark Furon). Uses PXL files. Allows inclusion of PostScript. Written in C. Source is included. Suppliers: Prime distribution tape.

- Symbolics Lisp

dvi-stream (by Chris Lindblad). Written in Zetalisp. Source is included. Uses the Generic Hardcopy Interface to drive the PostScript printer. Supports landscape printing and graphics inclusion. Suppliers: Massachusetts Institute of Technology.

- Unix

dvi2ps. Uses PXL files. Allows landscape printing and graphics inclusion. Written in C. Source is included. Suppliers: Massachusetts Institute of Technology.

dvi2ps (modified by Paul Leyland). Uses PK files. Suppliers: Oxford [1].

dvi2ps (modified by Piet van Oostrum). Uses PK files and PostScript built-in fonts. Suppliers: van Oostrum.

dvi3ps (by Kevin Coombes). Uses GF, PK or PXL files. Supports use of printer-resident fonts, Asian fonts with the technique specified by the Japan TEX Users Group, runtime settable printer resolution, PSFig specials, and page selection. Suppliers: FTP ⟨stag.math.lsa.umich.edu⟩.

DVIALW (by Nelson H. F. Beebe and Neal Holtz). See description above. Suppliers: FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩, University of Utah.

DVIlaser/PS. See description above. Available for DEC/Unix, Apollo and Sun. Cost: $500 workstations; $750 mainframes. Site licenses available. Academic discounts available. Suppliers: ArborText, Inc.

DVIPS (by Tom Rokicki). See description above. Suppliers: FTP ⟨labrea.stanford.edu⟩.

dvitops (by James Clark). See description above. Suppliers: Aston, Clark, FTP ⟨june.cs.washington.edu⟩, FTP ⟨uk.ac.aston.tex⟩, FTP ⟨ymir.claremont.edu⟩.

dvitps (by Stephan v. Bechtolsheim). Uses GF, PK, and PXL files. Fonts may have up to 256 characters. Allows use of printer-resident PostScript fonts. Supports inclusion of PostScript graphics (through PSFIG), graphics through the tpic specials, arbitrary extension and mapping of the PostScript fonts, memory management, and generation of TranScript-compatible code. Included are programs for creation of TFM files for PostScript fonts (pfd2tfm) and for management of PostScript font mappings (printpdr). Suppliers: Bechtolsheim, FTP ⟨cs.purdue.edu⟩, Unix distribution tape.

Unspecified program. Suppliers: Carleton University.

- VAX/VMS

DVIALW (by Nelson H. F. Beebe and Neal Holtz). See description above. Suppliers: FTP ⟨ctrsci.utah.edu⟩, University of Utah.

DVIlaser/PS. See description above. Cost: $500 workstations; $750 mainframes. Site licenses available. Academic discounts available. Suppliers: ArborText, Inc.

DVIOUT (by Scott Campbell). Uses GF, PK, and PXL files. Allows landscape printing, inclusion of MacDraw bitmaps, inclusion of Tektronix plot files, drawing of line, arc, point, and filled polygons through `\special` commands, and TEX–XET support. Written in C and Macro-32. The program comes with a well-featured PostScript symbiont. Suppliers: DECUS TEX Collection, FTP ⟨ymir.claremont.edu⟩.

DVIPS (by Tom Rokicki). See description above. Suppliers: DECUS TEX Collection.

Dvi/PS (by Alec Dunn). Uses GF, and both word- and byte-packed PXL files. Allows landscape printing, inclusion of PostScript graphics (described in *TUGboat* 8#2), and use of PostScript fonts. A PostScript-from-Mac program is available on request. The program communicates with the printer to determine what resolution/set of fonts to use. Written in Pascal. Source not included. Cost: $500. Suppliers: University of Sydney.

PSPRINT (by Andrew Trevorrow). Uses PK and PXL files. Allows landscape printing, inclusion of PostScript graphics and use of printer-resident fonts. Written in DCL and Modula-2. Source included. Suppliers: Aston, DECUS TEX Collection, FTP ⟨aston.ac.uk⟩, FTP ⟨ymir.claremont.edu⟩, INFN/CNAF.

T2/script. Uses GF or PK files. Allows landscape printing, use of built-in and downloadable PostScript fonts, inclusion of graphics (rotated to match surrounding text) and produces output conforming to Adobe Document Structuring Conventions v2.1. Cost: $495 (1600bpi magtape), $515 (TK50 cartridge). Suppliers: Northlake Software.

QMS Kiss, Smartwriter

- Amiga

Unspecified program. Uses PK files. Allows landscape printing. Suppliers: Radical Eye Software.

QMS Lasergrafix

dvi2qms (Chris Lindblad). Uses PXL files. Includes support for landscape printing and graphics inclusion. Can be run as a filter. Written in C. Source included.

GTEX. Uses PK files. This driver is part of a CGM interpreter package and shares output drivers with that package.

DVIlaser/QMS. Uses GF, PK, and PXL files. Allows inclusion of graphics, use of printer resident fonts, font substitution, font scaling, landscape printing, and magnifies or shrinks images. VF support included.

DVIQMS. Uses PXL files. Supports landscape printing.

- Amdahl MTS

DVIlaser/QMS. See description above. Supported on 'as is' basis. Cost: $750. Site licenses available. Academic discounts available. Suppliers: ArborText, Inc.

- Amiga

Unspecified program. Uses PK files. Allows landscape printing. Suppliers: Radical Eye Software.

- Data General MV

Unspecified program. Suppliers: Texas A&M [1].

- IBM MVS

DVIQMS. See description above. Suppliers: Gesellschaft für Mathematik und Datenverarbeitung [1].

- IBM PC

DVIlaser/QMS. See description above. Supported on 'as is' basis. Cost: $225. Site licenses available. Academic discounts available. Suppliers: ArborText, Inc., TEX Users Group.

- IBM VM/CMS

DVIlaser/QMS. See description above. Supported on 'as is' basis. Cost: $750. Site licenses available. Academic discounts available. Suppliers: ArborText, Inc.

DVIQMS. See description above. Suppliers: Gesellschaft für Mathematik und Datenverarbeitung [1].

- Prime

DVILG8, DVILG12, DVILG15 (by Norman Naugle). Uses GF, PXL, or PK files. Allows inclusion of QUIC commands from files or as part of the TEX input stream. Allows landscape printing through the `\special` command. Allows use of printer-resident fonts. Includes CRERES for creating printer-resident fonts. The Prime distribution tape includes pre-compiled copies of the programs. Written in WEB. Source included. Cost: $150 from n^2; if the program is obtained from the Prime distribution tape, it is considered shareware—sites using the program are encouraged to send a $150 contribution to n^2 Consultants. Suppliers: n^2 Consultants, Prime distribution tape.

- Siemens BS2000

DVIQMS. See description above. Suppliers: Gesellschaft für Mathematik und Datenverarbeitung [1].

- Symbolics Lisp

dvi-stream (by Chris Lindblad). Written in Zetalisp. Source is included. Uses the Generic Hardcopy Interface to drive the QMS Lasergrafix printer. Supports landscape printing and graphics inclusion. Suppliers: Massachusetts Institute of Technology.

- Unix

dvi2qms (Chris Lindblad). See description above. Suppliers: Massachusetts Institute of Technology.

GTEX. See description above. Suppliers: FTP ⟨casce.psc.edu⟩.

DVIlaser/QMS. See description above. Supported on 'as is' basis. Available for VAX Unix, Apollo and Sun. Cost: $500 workstations; $750 mainframes. Site licenses available. Academic discounts available. Suppliers: ArborText, Inc.

quicspool (Scott Simpson). Uses PK files. Supports landscape printing and all 4.2 BSD spooler functions for QMS QUIC printers. Also included are drivers for troff, programs to convert TFM to troff width tables, and METAFONT code for troff fonts. Written in C with lex and yacc. Source included. Suppliers: Unix distribution tape.

Unspecified program. Runs on Sun. Suppliers: University of Delaware.

Unspecified program. Runs on Apollo. Suppliers: Scan Laser.

Unspecified program. Runs on an HP 9000/500. Suppliers: Texas A&M [2].

- VAX/VMS

GTEX. See description above. Suppliers: FTP ⟨b.psc.edu⟩.

DVIlaser/QMS. See description above. Supported on 'as is' basis. Cost: $500 workstations; $750 mainframes. Site licenses available. Academic discounts available. Suppliers: ArborText, Inc.

Unspecified program. Suppliers: GA Technologies.

Unspecified program. Suppliers: n^2 Consultants.

Talaris (See also QMS Lasergrafix)

- IBM MVS

Unspecified program. Suppliers: Talaris.

- IBM PC

Unspecified program. Suppliers: Talaris.

- IBM VM/CMS

Unspecified program. Suppliers: Talaris.

- Unix

Unspecified program. Suppliers: Talaris.

- VAX/VMS

Unspecified program. Suppliers: Talaris.

Xerox 2700II, 3700, 4045

DVIX27 (by John Gourlay). Uses Xerox 2700 special fonts (the cm* fonts are supplied up to magstep 5 in this format). Written in WEB. Source included.

- CDC Cyber

Unspecified program. Suppliers: Bochum.

- DEC-20

DVIX27 (by John Gourlay). See description above. Suppliers: Xerox, Ohio State University.

- IBM VM/CMS

DVI2700 (by Maurice Vallino and Chantal Durand). Uses Xerox 2700 special fonts. Inclusion of Xerox bitmap files is made possible by the `\special` command. An auxiliary program, PXLXEROX, is provided to allow conversion of PXL files to Xerox 2700 format. Written in Pascal. Source included. Suppliers: Ecole Normale Superieure.

DVIX27 (by John Gourlay). See description above. Suppliers: Xerox.

- Unix

DVIX27 (by John Gourlay). See description above. Suppliers: Xerox.

- VAX/VMS

DVIX27 (by John Gourlay). See description above. Suppliers: Xerox.

Unspecified program. Suppliers: Brigham Young University.

Xerox 8700, 8790, 9700, 9790, 4050

DVIXER (by Paul Grosso). Written in WEB. Source included. Uses fonts preloaded onto the printer. Allows duplex printing.

TEXrox (by Thomas J. Reid). Written in C. Source included. Uses fonts preloaded onto the printer. Allows duplex printing and four basic page orientations plus special formats for booklets and reference cards. Multiple DVI files may be merged using the driver. Includes utilities for creating Xerox fonts from GF or PXL files and for creating TFM files for Xerox internal fonts.

- Amdahl MTS

DVIXER (by Paul Grosso). See description above. Supported on 'as is' basis. Cost: $1500. Site licenses available. Academic discounts available. Suppliers: ArborText, Inc.

DVIXER (by Paul Grosso; modified by Kari Gluski). See description above. Supported on 'as is' basis. Suppliers: University of Michigan.

• IBM MVS

DVIXER (by Paul Grosso). See description above. Supported on 'as is' basis. Cost: $1500. Site licenses available. Academic discounts available. Suppliers: ArborText, Inc.

TEXrox (by Thomas J. Reid). See description above. Written in C and 370 Assembler. Suppliers: Texas A&M [3].

• IBM VM/CMS

DVIXER (by Paul Grosso). See description above. Supported on 'as is' basis. Cost: $1500. Site licenses available. Academic discounts available. Suppliers: ArborText, Inc.

TEXrox (by Thomas J. Reid). Written in C. See description above. Suppliers: Texas A&M [3].

• Unix

TEXrox (by Thomas J. Reid). See description above. Tested under VM/UTS and Sun Unix. Suppliers: Texas A&M [3].

Unspecified program. Runs on Apollo. Suppliers: COS Information.

Unspecified program. Runs on Apollo. Suppliers: Scan Laser.

Unspecified program. Suppliers: University of Delaware.

• VAX/VMS

DVIXER (by Paul Grosso). See description above. Supported on 'as is' basis. Cost: $1500 mainframes, $1000 workstations. Site licenses available. Academic discounts available. Suppliers: ArborText, Inc.

TEXrox (by Thomas J. Reid). See description above. Suppliers: Texas A&M [3].

Unspecified program. Suppliers: Advanced Computer Communications.

Drivers for Impact Printers and Miscellaneous Output Devices

Apple ImageWriter

DVIM72, DVIMAC (by Nelson H. F. Beebe). Uses GF, PK, or PXL files. DVIM72 uses the Imagewriter at 72dpi, DVIMAC uses a resolution of 144dpi. Written in C.

• Acorn

DVIM72, DVIMAC (by Nelson H. F. Beebe). See description above. (Beta test version available on request.) Suppliers: University of Utah.

• Amiga

Unspecified program. Uses PK files. Suppliers: Radical Eye Software.

• Apple Macintosh

DVIM72-Mac 1.8 (by Jim Walker). Uses PK files. Can run in the background under Multifinder. Intended for use with OzTEX. Suppliers: FTP ⟨giza.cis.ohio-state.edu⟩.

• Atari ST

DVIM72, DVIMAC (by Nelson H. F. Beebe). See description above. Suppliers: FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩, University of Utah.

• DEC-20

DVIM72, DVIMAC (by Nelson H. F. Beebe). See description above. Suppliers: FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩.

• IBM PC

DVIDOT (by Eberhard Mattes). Uses PK or PXL files. DVIDOT is a generic dot matrix printer that supports different printers through a configuration file. Supports VF files. Suppliers: Aston, FTP ⟨rusmv1.rus.uni-stuttgart.de⟩, FTP ⟨terminator.cc.umich.edu⟩, FTP ⟨uk.ac.aston.tex⟩, Radel.

DVIM72, DVIMAC (by Nelson H. F. Beebe). See description above. Written in Microsoft C. Suppliers: Aston, FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩, University of Utah.

• IBM VM/CMS

DVIM72, DVIMAC (by Nelson H. F. Beebe). See description above. Written in Waterloo C. (Beta test version available on request.) Suppliers: University of Utah.

• Prime

DVIM72, DVIMAC (by Nelson H. F. Beebe). See description above. (Beta test version available on request.) Suppliers: University of Utah.

• Unix

DVIM72, DVIMAC (by Nelson H. F. Beebe). See description above. Runs on most Unix variants. Suppliers: FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩, University of Utah.

• VAX/VMS

DVIM72, DVIMAC (by Nelson H. F. Beebe). See description above. Suppliers: FTP ⟨ctrsci.utah.edu⟩, University of Utah.

Unspecified program. Suppliers: Louisiana State University.

Benson 9424

- IBM VM/CMS

DVIBENA3, DVIBENA4, DVIBENA5 (by Dr. Georg Bayer). Uses PXL files at 254dpi. DVIBENA3 creates a page for DIN A3 paper, DVIBENA4 creates a page for DIN A4 paper placing 2 pages per sheet, and DVIBENA5 creates a page for DIN A5 paper placing 4 pages per sheet. Suppliers: Technische Universität Braunschweig.

C. Itoh 8510A

- IBM PC

DVIDOT (by Eberhard Mattes). Uses PK or PXL files. DVIDOT is a generic dot matrix printer that supports different printers through a configuration file. Supports VF files. Suppliers: Aston, FTP ⟨rusmv1.rus.uni-stuttgart.de⟩, FTP ⟨terminator.cc.umich.edu⟩, FTP ⟨uk.ac.aston.tex⟩, Radel.

Citizen 120-D

- Amiga

Unspecified program. Uses PK files. Suppliers: Radical Eye Software.

DEC LA75, LP100

DVIL75 (by John Sauter). Uses GF, PK, and PXL files. Written in C.

- Acorn

DVIL75 (by John Sauter). See description above. (Beta test version available on request.) Suppliers: University of Utah.

- Atari ST

DVIL75 (by John Sauter). See description above. Suppliers: FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩, University of Utah.

- DEC-20

DVIL75 (by John Sauter). See description above. Suppliers: FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩.

DVILA (by John Gourlay). Uses PXL files. Comes with PXLPXL, a utility for converting PXL files from one resolution to another. Written in WEB. Source included. Suppliers: Ohio State University.

- IBM PC

DVIL75 (by John Sauter). See description above. Written in Microsoft C. Suppliers: FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩, University of Utah.

- IBM VM/CMS

DVIL75 (by John Sauter). See description above. Written in Waterloo C. (Beta test version available on request.) Suppliers: University of Utah.

- Prime

DVIL75 (by John Sauter). See description above. (Beta test version available on request.) Suppliers: University of Utah.

- Unix

DVIL75 (by John Sauter). See description above. Suppliers: FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩, University of Utah.

- VAX/VMS

DVIL75 (by John Sauter). See description above. Suppliers: FTP ⟨ctrsci.utah.edu⟩, University of Utah.

Epson FX/MX/JX/RX

DVIE72, DVIEPS (by Marcus Moehrman). Uses GF, PK, or PXL files. DVIE72 prints at $60h \times 72v$ resolution, DVIEPS prints at $240h \times 216v$ resolution. Written in C. Source is included.

- Acorn

DVIE72, DVIEPS (by Marcus Moehrman). See description above. (Beta test version available on request.) Suppliers: University of Utah.

- Amiga

Unspecified program. Uses PK files. Suppliers: Radical Eye Software.

- Atari ST

DVIE72, DVIEPS (by Marcus Moehrman). See description above. Suppliers: FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩, University of Utah.

Unspecified program. Suppliers: TEXsys.

Unspecified program. Suppliers: Tools GmbH Bonn.

- DEC-20

DVIE72, DVIEPS (by Marcus Moehrman). See description above. Suppliers: FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩.

- HP 1000

Unspecified program. Suppliers: JDJ Wordware.

- HP 3000

Unspecified program. Suppliers: University of Sheffield.

- IBM PC

DVIDOT (by Eberhard Mattes). Uses PK or PXL files. DVIDOT is a generic dot matrix printer that supports different printers through a configuration file. Supports VF files. Suppliers:

Aston, FTP ⟨rusmv1.rus.uni-stuttgart.de⟩, FTP ⟨terminator.cc.umich.edu⟩, FTP ⟨uk.ac.aston.tex⟩, Radel.

DVIE72, DVIEPS (by Marcus Moehrman). See description above. Written in Microsoft C. Suppliers: FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩, Radel, University of Utah.

DVIEPS (by Gavin Melville and Gordon Findlay from the Beebe driver). Uses GF, PK, or PXL files at $240h \times 216v$ resolution. Written in Microsoft C, no executables. The page bitmap is not held in RAM. Suppliers: Radel.

PC DOT Epson FX/RX. Cost: $95. Suppliers: Personal TEX.

Unspecified program. Suppliers: Texas A&M [1].

Unspecified program. Suppliers: Università Degli Studi Milan.

Unspecified program. Suppliers: University of Sheffield.

• IBM VM/CMS

DVIE72, DVIEPS (by Marcus Moehrman). See description above. Written in Microsoft C. (Beta test version available on request.) Suppliers: University of Utah.

• Prime

DVIE72, DVIEPS (by Marcus Moehrman). See description above. (Beta test version available on request.) Suppliers: University of Utah.

• Unix

DVIE72, DVIEPS (by Marcus Moehrman). See description above. Suppliers: FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩, University of Utah.

• VAX/VMS

DVIE72, DVIEPS (by Marcus Moehrman). See description above. Suppliers: FTP ⟨ctrsci.utah.edu⟩, University of Utah.

Epson LQ, NEC P6/P7

• Amiga

Unspecified program. Uses GF, PK or PXL files. Based on the Beebe 2.10 drivers. Suppliers: FTP ⟨tut.cis.ohio-state.edu⟩.

Unspecified program. Uses PK files. Suppliers: Radical Eye Software.

• Atari ST

Unspecified program. Suppliers: TEXsys.

• IBM PC

DVIDOT (by Eberhard Mattes). Uses PK or PXL files. DVIDOT is a generic dot matrix printer that supports different printers through a configuration file. Can print at 180 × 180, 360 × 180 or 360 × 360. Supports VF files. Suppliers: Aston, FTP ⟨rusmv1.rus.uni-stuttgart.de⟩, FTP ⟨terminator.cc.umich.edu⟩, FTP ⟨uk.ac.aston.tex⟩, Radel.

DVINECLQ (by Fuyun Ling from the Beebe DVITOS program). Uses GF, PK or PXL files at 360dpi. The page bitmap is swapped to ramdisk or hard disk, or sent directly to LPT1: Distributed as executables only. Suppliers: Channel 1 BBS, Ling, Radel.

PC DOT Epson LQ. Cost: $95. Suppliers: Personal TEX.

Fujitsu

• Atari ST

Unspecified program. Uses PK files. Suppliers: TEXsys.

• Cadmus 9200

Unspecified program. Suppliers: University of Köln.

GE 3000

• Unix

Unspecified program. Runs on Apollo. Suppliers: COS Information.

HP DeskJet

• Amiga

Unspecified program. Uses PK files. Suppliers: Radical Eye Software.

• Atari ST

Unspecified program. Uses GF, PK or PXL files. Based on Beebe 2.10 drivers. Comes in two versions for different memory configurations. Requires a hard disk. Suppliers: FTP ⟨terminator.cc.umich.edu⟩, Long.

• IBM PC

dvidjp (by Paul Kirkaas). Uses GF, PK or PXL files. Based on Beebe 2.10 drivers. Suppliers: FTP ⟨ymir.claremont.edu⟩.

Unspecified program. Cost: $119. Suppliers: Personal TEX.

Unspecified program. Cost: $100. Suppliers: The Toolsmith.

HP InkJet

• Amiga

Unspecified program. Uses PK files. Suppliers: Radical Eye Software.

MPI Sprinter

DVIMPI (by Nelson H. F. Beebe). Uses GF, PK, or PXL files. Written in C.

• Acorn

DVIMPI (by Nelson H. F. Beebe). See description above. (Beta test version available on request.) Suppliers: University of Utah.

• Atari ST

DVIMPI (by Nelson H. F. Beebe). See description above. Suppliers: FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩, University of Utah.

• DEC-20

DVIMPI (by Nelson H. F. Beebe). See description above. Suppliers: FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩.

• IBM PC

DVIMPI (by Nelson H. F. Beebe). See description above. Written in Microsoft C. Suppliers: FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩, University of Utah.

• IBM VM/CMS

DVIMPI (by Nelson H. F. Beebe). See description above. Written in Waterloo C. (Beta test version available on request.) Suppliers: University of Utah.

• Prime

DVIMPI (by Nelson H. F. Beebe). See description above. (Beta test version available on request.) Suppliers: University of Utah.

• Unix

DVIMPI (by Nelson H. F. Beebe). See description above. Suppliers: FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩, University of Utah.

• VAX/VMS

DVIMPI (by Nelson H. F. Beebe). See description above. Suppliers: FTP ⟨ctrsci.utah.edu⟩, University of Utah.

NDK Printstar

• IBM VM/CMS

DVINDKN, DVINDKQ (by Dr. Georg Bayer). Uses PXL files at 120dpi. DVINDKN prints lines of text horizontally, while DVINDKQ prints lines of text vertically. Suppliers: Technische Universität Braunschweig.

Okidata

DVIO72, DVIOKI (by Nelson H. F. Beebe). Uses GF, PK, or PXL files. DVIO72 prints at 72dpi resolution; DVIOKI prints at 144dpi. Written in C.

• Acorn

DVIO72, DVIOKI (by Nelson H. F. Beebe). See description above. (Beta test version available on request.) Suppliers: University of Utah.

• Amiga

Unspecified program. Uses PK files. Suppliers: Radical Eye Software.

• Atari ST

DVIO72, DVIOKI (by Nelson H. F. Beebe). See description above. Suppliers: FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩, University of Utah.

• DEC-20

DVIO72, DVIOKI (by Nelson H. F. Beebe). See description above. Suppliers: FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩.

• IBM PC

DVIO72, DVIOKI (by Nelson H. F. Beebe). See description above. Written in Microsoft C. Suppliers: FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩, University of Utah.

• IBM VM/CMS

DVIO72, DVIOKI (by Nelson H. F. Beebe). See description above. Written in Waterloo C. (Beta test version available on request.) Suppliers: University of Utah.

• Prime

DVIO72, DVIOKI (by Nelson H. F. Beebe). See description above. (Beta test version available on request.) Suppliers: University of Utah.

• Unix

DVIO72, DVIOKI (by Nelson H. F. Beebe). See description above. Suppliers: FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩, University of Utah.

• VAX/VMS

DVIO72, DVIOKI (by Nelson H. F. Beebe). See description above. Suppliers: FTP ⟨ctrsci.utah.edu⟩, University of Utah.

Printronix

DVIPRX (by Nelson H. F. Beebe). Uses GF, PK, or PXL files. Written in C.

• Acorn

DVIPRX (by Nelson H. F. Beebe). See description above. (Beta test version available on request.) Suppliers: University of Utah.

- Atari ST

DVIPRX (by Nelson H. F. Beebe). See description above. Suppliers: FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩, University of Utah.

- Data General MV

Unspecified program. Suppliers: Texas A&M [1].

- DEC-20

DVIPRX (by Nelson H. F. Beebe). See description above. Suppliers: FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩.

- IBM PC

DVIPRX (by Nelson H. F. Beebe). See description above. Written in Microsoft C. Suppliers: FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩, University of Utah.

Unspecified program. Suppliers: Texas A&M [1].

- IBM VM/CMS

DVIPRX (by Nelson H. F. Beebe). See description above. Written in Waterloo C. (Beta test version available on request.) Suppliers: University of Utah.

- Prime

DVIPRX (by Nelson H. F. Beebe). See description above. (Beta test version available on request.) Suppliers: University of Utah.

- Unix

DVIPRX (by Nelson H. F. Beebe). See description above. Suppliers: FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩, University of Utah.

- VAX/VMS

DVIPRX (by Nelson H. F. Beebe). See description above. Suppliers: FTP ⟨ctrsci.utah.edu⟩, University of Utah.

Texas Instruments 855

- IBM PC

Unspecified program. Suppliers: Texas A&M [1].

Toshiba

DVITOS (by Nelson H. F. Beebe). Uses GF, PK, or PXL files. Written in C.

- Acorn

DVITOS (by Nelson H. F. Beebe). See description above. (Beta test version available on request.) Suppliers: University of Utah.

- Atari ST

DVITOS (by Nelson H. F. Beebe). See description above. Suppliers: FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩, University of Utah.

- Data General MV

Unspecified program. Suppliers: Texas A&M [1].

- DEC-20

DVITOS (by Nelson H. F. Beebe). See description above. Suppliers: FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩.

- IBM PC

DVITOS (by Nelson H. F. Beebe). See description above. Written in Microsoft C. Suppliers: FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩, University of Utah.

PC DOT Toshiba. Cost: $95. Suppliers: Personal TEX.

- IBM VM/CMS

DVITOS (by Nelson H. F. Beebe). See description above. Written in Waterloo C. (Beta test version available on request.) Suppliers: University of Utah.

- Prime

DVITOS (by Nelson H. F. Beebe). See description above. (Beta test version available on request.) Suppliers: University of Utah.

- Unix

DVITOS (by Nelson H. F. Beebe). See description above. Suppliers: FTP ⟨science.utah.edu⟩, FTP ⟨ymir.claremont.edu⟩, University of Utah.

- VAX/VMS

DVITOS (by Nelson H. F. Beebe). See description above. Suppliers: FTP ⟨ctrsci.utah.edu⟩, University of Utah.

Unspecified program. Suppliers: Procyon Informatics.

Varian

- VAX/VMS

Unspecified program. Suppliers: Science Applications.

Versatec

- IBM MVS

DVIVER. Uses PXL files. Suppliers: Gesellschaft für Mathematik und Datenverarbeitung [1].

- IBM VM/CMS

DVI82 (by Yossie Silverman). Uses PXL files. Allows inclusion of raster files. Written in assembly language. Source is included. Suppliers: Weizmann Institute.

- Unix

Unspecified program. This program is included with iptex. Suppliers: University of Maryland.

Miscellaneous (generic print interfaces, special graphic formats, etc.)

- IBM PC

DVIDOT (by Eberhard Mattes). Uses PK or PXL files. DVIDOT is a generic dot matrix printer that supports different printers through a configuration file. Supports VF files. Suppliers: Aston, FTP ⟨rusmv1.rus.uni-stuttgart.de⟩, FTP ⟨terminator.cc.umich.edu⟩, FTP ⟨uk.ac.aston.tex⟩, Radel.

DVIMSP (by Eberhard Mattes). Uses PK or PXL files. Creates Microsoft Paint files. Supports VF files. Suppliers: Aston, FTP ⟨rusmv1.rus.uni-stuttgart.de⟩, FTP ⟨terminator.cc.umich.edu⟩, FTP ⟨uk.ac.aston.tex⟩, Radel.

TEX-FAX. Sends DVI files to a FAX machine. Cost: $395 (for 4800bps FAX board), $795 (for 9600 bps FAX board). Suppliers: Kinch Computing.

Drivers for Phototypesetters

Allied Linotype CRTronic

- VAX/VMS

Unspecified program. Suppliers: Procyon Informatics.

Allied Linotype L100, L300P

- IBM PC

Unspecified program. Suppliers: Personal TEX.

Allied Linotype L202

- IBM PC

Unspecified program. Suppliers: Personal TEX.

- VAX/VMS

Unspecified program. Suppliers: Procyon Informatics.

Autologic APS-5, Micro-5

DVIAPS. Autologic resident fonts and logo processing are supported, and a separate program that creates laser printer and screen resolution fonts from Autologic font tapes is available. VF support is provided.

- IBM PC

DVIAPS. See description above. Cost: $3000. Site licenses available. Academic discounts available. Suppliers: ArborText, Inc.

- Unix

DVIAPS. See description above. Available for VAX Unix and Sun. Cost: $3000. Site licenses available. Academic discounts available. Suppliers: ArborText, Inc.

Unspecified program. Runs on Apollo. Suppliers: COS Information.

Unspecified program. Runs on Apollo. Suppliers: Scan Laser.

- VAX/VMS

DVIAPS. See description above. Cost: $3000. Site licenses available. Academic discounts available. Suppliers: ArborText, Inc.

Unspecified program. Suppliers: Intergraph.

Compugraphic 8400

- HP 3000

Unspecified program. Suppliers: University of Sheffield.

- IBM PC

DVICG. VF support is provided. Cost: $2000. Site licenses available. Academic discounts available. Suppliers: ArborText, Inc.

- Unix

DVICG. VF support is provided. Site licenses available. Academic discounts available. Suppliers: ArborText, Inc.

- VAX/VMS

CGTEX. Includes FDtoPL for creating TFM files for Compugraphic fonts as well as some pre-generated TFMs for selected Compugraphic fonts. Cost: $3400 on 600′ 1600bpi magtape. Suppliers: Northlake Software.

Compugraphic 8600

- CDC Cyber

Unspecified program. Suppliers: Aarhus University.

- IBM PC

DVICG. VF support is provided. Cost: $2000. Site licenses available. Academic discounts available. Suppliers: ArborText, Inc.

- IBM VM/CMS

Unspecified program. Written in WEB. Source included. Suppliers: FTP ⟨ymir.claremont.edu⟩, Washington State University.

- Sperry 1100

Unspecified program. Suppliers: University of Wisconsin.

- Unix

DVICG. VF support is provided. Site licenses available. Academic discounts available. Suppliers: ArborText, Inc.

• VAX/VMS

CGTEX. Includes FDtoPL for creating TFM files for Compugraphic fonts as well as some pre-generated TFMs for selected Compugraphic fonts. Cost: $3400 on 600′ 1600bpi magtape. Suppliers: Northlake Software.

Compugraphic 8800

• IBM PC

DVICG. VF support is provided. Cost: $2000. Site licenses available. Academic discounts available. Suppliers: ArborText, Inc.

• Unix

DVICG. VF support is provided. Site licenses available. Academic discounts available. Suppliers: ArborText, Inc.

Harris 7500

• Unix

Unspecified program. Suppliers: SARA.

Hell Digiset

DVIDIGI. Uses special format files. A program for converting PXL files to this format is included.

• IBM MVS

DVIDIGI. See description above. Suppliers: Gesellschaft für Mathematik und Datenverarbeitung [2].

• Siemens BS2000

DVIDIGI. See description above. Suppliers: Gesellschaft für Mathematik und Datenverarbeitung [2].

Screen Previewers

Amiga

Unspecified program. Uses PK files. Written in C. Included with AmigaTEX. Suppliers: Radical Eye Software.

Apollo

DVIAPOLLO (by Leonard N. Zubkoff). Supports GPR. Uses Apollo font files. Included is a program to convert PXL files at 118 dpi to Apollo font files. Suppliers: FTP ⟨june.cs.washington.edu⟩.

Preview. Uses PXL, GF, and PK files as well as tuned PostScript fonts (the base set available with PostScript printers). Features include font substitution, page magnification and shrinking, searching for character strings, selection of arbitrary pages, display of pages in two-up mode, and preview of integrated bitmap graphics. Cost: $500. Suppliers: ArborText, Inc.

Texx (by Dirk Grunwald). Supports X-11 Windows System. Uses PK, GF, and PXL files at output device resolution. The window size may be changed for closeups of the page. Two pages may be viewed simultaneously. Suppliers: FTP ⟨cs.uiuc.edu⟩.

Atari ST

DVIST (by Avy Moise and Tyler Ivanco). Suppliers: FTP ⟨ssyx.ucsc.edu⟩.

Unspecified program. Uses PK files. Suppliers: TEXsys.

Unspecified program. Suppliers: Tools GmbH Bonn.

Cadmus 9200

Unspecified program. Suppliers: University of Köln.

Data General MV

Unspecified program. Suppliers: Texas A&M [1].

DEC Rainbow PC100

RBDVI 1.0. Uses a limited set of 19 CM and 5 LATEX fonts built into the program for efficiency's sake. Uses font substitution to display other fonts. Fits in 150K of disk space. Cost: $59.95. Special discount available for IRUG members. Suppliers: SullivanSFT.

DEC-20

DVIBIT (by Stephan Bechtolsheim, Bob Brown, Robert Wells, Jim Schaad, Richard Furuta, Nelson H. F. Beebe, Simon Barnes, Robin Rohlicek). Supports BBN Bitgraph Terminal. Uses GF, PK, or PXL files. Written in C. Source included. Suppliers: University of Utah.

DVIDOC (by John Gourlay). Supports ASCII output. Reads font information from TFM files and generates a text file representation of the DVI file. Suppliers: Ohio State University.

DEC RISC Ultrix

Preview. Uses PXL, GF, and PK files as well as tuned PostScript fonts (the base set available with PostScript printers). Features include font substitution, page magnification and shrinking, searching for character strings, selection of arbitrary pages, display of pages in two-up mode, and preview of integrated bitmap graphics. Suppliers: ArborText, Inc.

HP9000/500

DVIBIT (by Stephan Bechtolsheim, Bob Brown, Robert Wells, Jim Schaad, Richard Furuta, Nelson H. F. Beebe, Simon Barnes, Robin Rohlicek).

Supports BBN Bitgraph Terminal. Uses GF, PK, or PXL files. Written in C. Source included. Suppliers: University of Utah.

Preview. Uses PXL, GF, and PK files as well as tuned PostScript fonts (the base set available with PostScript printers). Features include font substitution, page magnification and shrinking, searching for character strings, selection of arbitrary pages, display of pages in two-up mode, and preview of integrated bitmap graphics. Suppliers: ArborText, Inc.

IBM MVS

DVIGDDM. Supports GDDM supported IBM display stations (including IBM 3179, 3192, 3193, and 3279). Uses PXL files. Suppliers: Gesellschaft für Mathematik und Datenverarbeitung [1].

Unspecified program. Supports Tektronix 4014 terminal. Suppliers: Università Degli Studi Milan.

IBM PC

cdvi. Supports EGA, CGA, Hercules. Cost: $175. Suppliers: n^2 Consultants.

CDVI 1.2 (by W.G. Sullivan). Supports VGA, MCGA, EGA, CGA, Hercules Monochrome, Olivetti. Uses the basic 16 plain TEX fonts in an internal format (they are part of the program). Cannot preview documents that contain fonts other than those 16. Suppliers: DECUS TEX Collection, Radel.

CDVI 2.02. Supports CGA, EGA, VGA, MCGA, Hercules, Olivetti-ATT *or* Toshiba 3100. Uses a limited set of 19 CM and 5 LATEX fonts built into the program for efficiency's sake. Uses font substitution to display other fonts. Fits in 150K of disk space. Note that there are separate programs for each display type which must be purchased separately. Cost: $35. Volume discounts available. Suppliers: SullivanSFT.

DVIEW. Supports CGA, EGA, VGA, [Hercules]. Uses PXL files. Hercules previewing is done through the SIMCGA program which is also included. Suppliers: DECUS TEX Collection, Radel.

dvimswin (by Doug McDonald). Supports Microsoft Windows. Suppliers: FTP ⟨wsmr-simtel20.army.mil⟩.

DVISCR (by Eberhard Mattes). Supports CGA, EGA, VGA, Hercules. Uses PK or PXL files. Supports VF files. Suppliers: Aston, FTP ⟨rusmv1.rus.uni-stuttgart.de⟩, FTP ⟨terminator.cc.umich.edu⟩, FTP ⟨uk.ac.aston.tex⟩, Radel.

DVIVGA 2.10 (by Doug McDonald from the Beebe drivers). Supports VGA. Uses GF, PK, or PXL files. Diff files from the Beebe driver source are supplied in lieu of the original C code. An executable is also included. Suppliers: Channel 1 BBS, DECUS TEX Collection, FTP ⟨wsmr-simtel20.army.mil⟩, Radel.

DVIVIK (by Eberhard Mattes). Supports Viking I. Uses PK or PXL files. Supports VF files. Suppliers: Aston, FTP ⟨rusmv1.rus.uni-stuttgart.de⟩, FTP ⟨terminator.cc.umich.edu⟩, FTP ⟨uk.ac.aston.tex⟩, Radel.

Maxview. Supports IBM EGA (mono or color), CGA, VGA, Hercules Graphics Card, Wyse WY/700, Genius VHR Full Page Display, AT&T 6300. Uses fonts from the laser printer driver in PK or PXL format to display text. Magnification may be set on entry. Suppliers: Aurion Tecnología, Personal TEX.

Preview. Supports IBM EGA (mono or color), MCGA, VGA, Hercules Graphics Card, Olivetti Monochrome, Tecmar Graphics Master, Genius VHR Full Page Display, ETAP Neftis Monitor, Toshiba 3100, AT&T 6300. Uses PXL, GF, and PK files as well as tuned PostScript fonts (the base set available with PostScript printers). Features include font substitution, page magnification and shrinking, searching for character strings, selection of arbitrary pages, display of pages in two-up mode, and preview of integrated bitmap graphics. Cost: $149. Suppliers: ArborText, Inc., TEX Users Group.

PTIVIEW. Uses GF, PK, and PXL files. On the fly magnification, on the fly inclusion of DVI files, font substitution, and 256 character fonts are supported. Cost: $149. Suppliers: Personal TEX.

TEXView 2.06 (by Laurie Benfield from the Beebe drivers). Supports Hercules. Uses GF, PK, or PXL files at 300dpi. The DVI file is first converted into a graphics image by DVIHERC then viewed by TEXView. Suppliers: Radel.

IBM PC/RT

Texx (by Dirk Grunwald). Supports X-11 Windows System. Uses PK, GF, and PXL files. The high resolution fonts used by laser printer drivers are used and shrunk to the screen resolution. The window size may be changed for closeups of the page. Two pages may be viewed simultaneously. Suppliers: FTP ⟨cs.uiuc.edu⟩.

IBM VM/CMS

DVI3279 (by Dr. Georg Bayer). Supports IBM 3179g and 3279 GDDM-driven graphics terminals. Uses PXL files at 120dpi. Written in WEB. Source and executables are included. Displays page either in 8 parts at natural size or 3 parts compressed (with a loss of readability). This driver is on CMS TEX tapes created after 3/88. Suppliers: Stanford CMS distribution tape, Technische Universität Braunschweig, Washington State University.

DVI82 (by Yossie Silverman). Supports IBM 3279, 3179-G terminals. Uses PXL files. Allows inclusion of raster files. Written in assembly language. Source is included. Suppliers: Weizmann Institute.

DVIGDDM. Supports GDDM supported IBM display stations (including IBM 3179, 3192, 3193, and 3279. Uses PXL files. Suppliers: Gesellschaft für Mathematik und Datenverarbeitung [1].

DVIview (by Don Hosek). Supports VT640-compatible terminals and Tektronix-compatible terminals connected through a Series-1/7171 protocol converter, IBM 3179g and 3279 GDDM-driven graphics terminals. Uses PK files. Allows resizing of preview "window" on the page and box outlines of characters for a quick view of page layout. Written in WEB. Source and executables are included. The previewer may be obtained by sending a blank tape and a check or money order for $30 to cover duplication costs to Don Hosek. Suppliers: Quixote.

DVIBIT (by Stephan Bechtolsheim, Bob Brown, Robert Wells, Jim Schaad, Richard Furuta, Nelson H. F. Beebe, Simon Barnes, Robin Rohlicek). Supports BBN Bitgraph Terminal. Uses GF, PK, or PXL files. Suppliers: University of Utah.

Sun Workstation

Preview. Uses PXL, GF, and PK files as well as tuned PostScript fonts (the base set available with PostScript printers). Features include font substitution, page magnification and shrinking, searching for character strings, selection of arbitrary pages, display of pages in two-up mode, and preview of integrated bitmap graphics. Cost: $500. Suppliers: ArborText, Inc.

TEXsun (by Dirk Grunwald). Supports Sunview Window System. Uses PK, GF, and PXL files. The high resolution fonts used by laser printer drivers are used and shrunk to the screen resolution. The window size may be changed for closeups of the page. Two pages may be viewed simultaneously. Suppliers: FTP ⟨cs.uiuc.edu⟩.

Texx (by Dirk Grunwald). Supports X-11 Windows System. Uses PK, GF, and PXL files. The high resolution fonts used by laser printer drivers are used and shrunk to the screen resolution. The window size may be changed for closeups of the page. Two pages may be viewed simultaneously. Suppliers: FTP ⟨cs.uiuc.edu⟩.

Unspecified program. Suppliers: University of California, Berkeley.

Unspecified program. Suppliers: University of California, Irvine.

Unix

DVIBIT (by Stephan Bechtolsheim, Bob Brown, Robert Wells, Jim Schaad, Richard Furuta, Nelson H. F. Beebe, Simon Barnes, Robin Rohlicek). Supports BBN Bitgraph Terminal. Uses GF, PK, or PXL files. Written in C. Source included. Suppliers: University of Utah.

DVIDMD (by Lou Salkind; portions adapted from Chris Torek's ctex package). Supports DMD 5620. Uses 118dpi fonts in GF, PK or PXL format. The program consists of two parts: a program that runs on the host computer and a program that is downloaded to the terminal. TPIC output is supported. The DMD terminal may be used in either window (MPX) or standalone mode. Suppliers: FTP ⟨nyu.edu⟩, Unix TEX distribution.

GTEX. Supports Tektronix, X11, Sun CGI. Uses PK files. This driver is part of a CGM interpreter package and shares output drivers with that package. Suppliers: FTP ⟨casce.psc.edu⟩.

SeeTEX 2.15 (by Dirk Grunwald). Supports X11. Supports display PostScript under DECwindows. Suppliers: FTP ⟨expo.lcs.mit.edu⟩, FTP ⟨foobar.colorado.edu⟩.

VAX/VMS

DVIBIT (by Stephan Bechtolsheim, Bob Brown, Robert Wells, Jim Schaad, Richard Furuta, Nelson H. F. Beebe, Simon Barnes, Robin Rohlicek). Supports BBN Bitgraph Terminal. Uses GF, PK, or PXL files. Written in C. Source included. Suppliers: University of Utah.

DVIOUT(by Scott Campbell). Supports Tektronix 4014. Uses GF, PK, and PXL files. Allows landscape printing, inclusion of MacDraw bitmaps, inclusion of Tektronix plot files, drawing of line, arc, point, and filled polygons through `\special` commands, and TEX–XET support. Written in C and Macro-32. Suppliers: DECUS TEX Collection, FTP ⟨ymir.claremont.edu⟩.

DVItoVDU (by Andrew Trevorrow). Supports AED 512, ANSI-compatible, DEC ReGIS, DEC VT100, DEC VT220, Tektronix 4014, Visual 500, 550. Uses PK or PXL files at output device resolution. Written in Modula 2. Source included. Suppliers: Aston, DECUS TEX Collection, FTP ⟨uk.ac.aston.tex⟩, FTP ⟨ymir.claremont.edu⟩, INFN/CNAF, Northlake Software.

DVIVIEW (by Peter Scott). Supports Tektronix, Pericom, Navplot, VT100, VT220, RAMTEK. Uses modified Hershey fonts for fast previewing. Suppliers: JPL.

GTEX. Supports Tektronix, GPX/UIS. Uses PK files. This driver is part of a CGM interpreter package and shares output drivers with that package. Suppliers: FTP ⟨b.psc.edu⟩.

PreTEXt. Supports Talaris T7600. The T7600 terminal has 28 resident CM fonts for previewing. Cost: $750. Suppliers: Talaris.

Preview. Uses PXL, GF, and PK files as well as tuned PostScript fonts (the base set available with PostScript printers). Features include font substitution, page magnification and shrinking, searching for character strings, selection of arbitrary pages, display of pages in two-up mode, and preview of integrated bitmap graphics. Runs under DEC-Windows. Suppliers: ArborText, Inc.

TP. Supports Tektronix 4010/4014 compatible. Uses special "stick-figure" fonts to display TEX output. Written in Fortran. Source included. Distributed on your choice of IBM 5.25″ disk, IBM 3.5″ disk, Mac 3.5″ disk, or TK50 cartridge (enclose your own or add $40 to order). Cost: $185. Suppliers: TPSoftware.

TXMAPPER, TXREGIS. Supports DEC ReGIS. Uses PXL files. Written in Fortran. Source included. Suppliers: Aston, DECUS TEX Collection, FTP ⟨uk.ac.aston.tex⟩, INFN/CNAF.

Vaxstation/Unix

Texx (by Dirk Grunwald). Supports X-11 Windows System. Uses PK, GF, and PXL files. The high resolution fonts used by laser printer drivers are used and shrunk to the screen resolution. The window size may be changed for closeups of the page. Two pages may be viewed simultaneously. This implementation may encounter some byte order problems. Suppliers: FTP ⟨cs.uiuc.edu⟩.

Vaxstation/VMS

DVIDIS (by Jerry Leichter). Supports GPX(UIS). Uses PK files at 150dpi (optional) and 300dpi subsampled to screen resolution. Distributed as executables only (source available only on special request). Suppliers: DECUS TEX Collection, Yale University.

Preview. Supports DECwindows. Uses PXL, GF, and PK files as well as tuned PostScript fonts (the base set available with PostScript printers), Features include font substitution, page magnification and shrinking, searching for character strings, selection of arbitrary pages, display of pages in two-up mode, and preview of integrated bitmap graphics. Cost: $500. Suppliers: ArborText, Inc.

Preview (by Randy Buckland). Supports GPX(UIS). Uses PK files at 78dpi. Allows magnification of preview window by TEX magsteps. Includes Font, a program for viewing fonts on the VAXstation display. Written in Ada. Source included. Suppliers: Research Triangle Institute.

T2/View. Supports DECwindows. Suppliers: Northlake Software.

TP. Supports GPX(UIS). Uses PK or PXL files at output driver resolution. Written in Fortran. Source included. Distributed on your choice of IBM 5.25″ disk, IBM 3.5″ disk, Mac 3.5″ disk, or TK50 cartridge (enclose your own or add $40 to order). Cost: $185. Suppliers: TPSoftware.

TXUIS. Supports GPX(UIS). Uses PXL files. Written in Fortran. Source included. Suppliers: INFN/CNAF.

XDVI. Supports DECwindows. Suppliers: DECUS TEX Collection, FTP ⟨ymir.claremont.edu⟩.

Unspecified program. Suppliers: Philips Kommunikations Industrie AG.

Supplier Information

- **Aarhus University**
- **ArborText, Inc.**

  Contact: Sales Department. 535 W. William Street, Suite 300, Ann Arbor, MI 48103. Internet: `sales@arbortext.com`. 313-996-3566.

- **Aston**

  Contact: Peter Abbott. Janet: `abbottp@uk.ac.aston`. Computing Service, Aston Triangle, Birmingham B4 7ET. A complete copy of the Aston archive is available in VMS BACKUP format on two 2400′ tapes at 6250bpi. To receive it send two blank tapes and return postage. The Unix TEX distribution may also be obtained in exchange for one 2400′ tape and return postage.

- **Aurion Tecnología**

  Contact: Armando Jinich. Arquímedes #3, 501, Polanco 11570, México, D.F. 905-545-7315. Telex: 171314 NACOME.

- **Bechtolsheim**

  Contact: Stephen v. Bechtolsheim. 2119 Old Oak Drive, West Lafayette, IN 47906. Send $80 ($110 outside the continental U.S.) and you will receive a Sun cartridge plus complete documentation.

- **Bochum**

  Contact: Norbert Schwartz. Ruhr Universität Bochum. 49 234 700-4014.

- **Brigham Young University**

  Contact: Paul Malquist. Internet: `malquistp@yvax.byu.edu`.

- **Canon**
- **Carleton University**

  Contact: Neal Holtz. 613-231-7145.

- **Channel 1 BBS**

  Modem: 617-354-8873.

- **Clark**
  Contact: James Clark. 30 Peel Street, London W8 7PD England. UUCP: `jjc@jclark.uucp`.
- **Columbia University**
  Contact: Frank da Cruz. Center for Computing Activities, Columbia University, 612 West 115th Street, New York, NY 10025. 212-280-5126.
- **COS Information**
  Contact: Gilbert Gingras. 5647 rue Ferrier, Montreal H4P 1N1, Quebec Canada. 514-738-2191.
- **CUBE Software**
  Contact: Warren W. Wolfe. 3002 Cadboro Bay Road, Victoria V8R 5J9, B.C. Canada. 604-380-4592.
- **DECUS TEX Collection**
  Library Order Processing, 219 Boston Post Road, BPO2, Marlboro, MA 01752. 508-480-3418, 508-480-3659, 508-480-3446.
- **Digital Equipment Corporation**
  Contact: John Sauter. 801128 Bates Road, Merrimack, NH 03054. Internet: `Sauter%Dssdev.DEC@Decwrl.DEC.com`. 603-881-2301.
- **Ecole Normale Superieure**
  Contact: Chantal Durand. Centre de Calcul, Ecole Normale Superieure, 45 rue d'Ulm, 75005 Paris, France.
- **FTP ⟨b.psc.edu⟩**
  Contact: Anjana Kar. Internet: `kar@b.psc.edu`. The GTEX files for VMS are located in `TEX$ROOT:[GPLOT]`.
- **FTP ⟨casce.psc.edu⟩**
  Contact: Anjana Kar. Internet: `kar@b.psc.edu`.
- **FTP ⟨cs.purdue.edu⟩**
  Contact: Stephan Bechtolsheim. `pub/svb/TeXPS/TEXPS-3.0.tar.Z` contains the dvitps driver.
- **FTP ⟨cs.uiuc.edu⟩**
  Contact: Dirk Grunwald. Internet: `grunwald@m.cs.uiuc.edu`. `pub/TeX/uiuctex2.0.tar.Z` contains Grunwald's drivers.
- **FTP ⟨csseq.tamu.edu⟩**
- **FTP ⟨ctrsci.utah.edu⟩**
  Contact: Nelson H. F. Beebe. 801-581-5254. Internet: `Beebe@Science.Utah.edu`.
- **FTP ⟨expo.lcs.mit.edu⟩**
- **FTP ⟨foobar.colorado.edu⟩**
- **FTP ⟨giza.cis.ohio-state.edu⟩**
  `pub/oztex/other` contains DVIM72-Mac.
- **FTP ⟨june.cs.washington.edu⟩**
  Contact: Elisabet Tachikawa. 206-543-6259. Internet: `elisabet@max.u.washington.edu`. `tex/dviapollo.tar.Z` (DVIAPOLLO in compressed tar file).
- **FTP ⟨nyu.edu⟩**
  Contact: Lou Salkind. Internet: `Salkind@Acf8.NYU.edu`. `pub/dmd.tar.Z` (DVIDMD in compressed tar file).
- **FTP ⟨orc.olivetti.com⟩**
  `pub/tmp/dvi2lj1_11.tar.Z` contains dvi2lj.
- **FTP ⟨rusmv1.rus.uni-stuttgart.de⟩**
  Eberhard Mattes' drivers are distributed with emTEX in `soft/tex/emtex`.
- **FTP ⟨science.utah.edu⟩**
  Contact: Nelson H. F. Beebe. (801) 581-5254. Internet: `Beebe@science.utah.edu`. `~ftp/pub/tex/pub/dvi/* and ~ftp/pub/tex/pub/dvi/doc/*` (Beebe drivers and documentation).
- **FTP ⟨ssyx.ucsc.edu⟩**
- **FTP ⟨stag.math.lsa.umich.edu⟩**
  Contact: Kevin Coombes. Internet: `kevin@math.lsa.umich.edu`. `pub/kevin/dvi3ps.tar.Z` (dvi3ps driver).
- **FTP ⟨terminator.cc.umich.edu⟩**
  The Atari ST DeskJet driver is in `/atari/tex`. Eberhard Mattes' drivers are distributed with the emTEX package in `msdos/text-mgmt/TeX/emtex`.
- **FTP ⟨tut.cis.ohio-state.edu⟩**
  The 24-pin Epson driver for the Amiga is in `pub/amigo`.
- **FTP ⟨uk.ac.aston.tex⟩**
  Contact: Peter Abbott. Janet: `abbottp@uk.ac.aston`. Accessible to JANET users only. Use Username PUBLIC, Password PUBLIC. For information on Mail access send a mail message with 3 lines containing (1) three hyphens, (2) your return mail address, and (3) the word HELP, to `texserver@uk.ac.aston`. Gustav Neumann's DVI2XX is available in `[TEX-ARCHIVE.DRIVERS.NEUMANN]` as `NEUMANN.BOO`. Nelson Beebe's drivers are in `[TEX-ARCHIVE.DRIVERS.BEEBE]`.

- **FTP ⟨wsmr-simtel20.army.mil⟩**
- **FTP ⟨ymir.claremont.edu⟩**

  Contact: Don Hosek. Internet: `dhosek@ymir.claremont.edu`. Bitnet: `dhosek@hmcvax.bitnet`. Get the file `[anonymous.tex]00readme.txt` before attempting to retrieve files. Text files from this site are also available by requesting files from `mailserv@ymir.claremont.edu`. Send the command help to that address for more details. `[anonymous.tex.drivers.beebe2_10...]` contains the Beebe drivers. `[anonymous.tex.drivers.beebe_extensions]` contains extensions to the Beebe driver family.
- **GA Technologies**
- **Gesellschaft für Mathematik und Datenverarbeitung [1]**

  Contact: Ferdinand Hommes. GMD Z1.BN, Postfach 1240, D-5205 St. Augustin 1, Federal Republic of Germany. Bitnet: `GRZTEX@DBNGMD21`. +49-228-8199621.
- **Gesellschaft für Mathematik und Datenverarbeitung [2]**

  Contact: Dr. Wolfgang Appelt. Schloss Birlinghoven - PF 1240, D-5202 St. Augustin 1, Federal Republic of Germany. UUCP: `seismo!unido!gmdzi!zi.gmd.dbp.de!appelt`.
- **Hewlett-Packard**

  Contact: Stuart Beatty. Hewlett-Packard Company, 3404 E. Harmony Rd., Ft. Collins, CO 80525. 303-229-2067.
- **INFN/CNAF**

  Contact: Maria Luisa Luvisetto. Via Mazzini 2, 40138 Bologna, Italy. 51-498286. Bitnet: `Miltex@Iboinfn`. DECnet: `<39947::luvisetto>`. The files are available only via DECnet/Span; for more information on this, send mail to the DECnet address: `<39947::luvisetto>`. No tape distribution from INFN/CNAF is available.
- **Intergraph**

  Contact: Mike Cunningham. One Madison Industrial Park, MS HQ1200, Huntsville, AL 35807. 205-772-2000.
- **JDJ Wordware**

  Contact: John D. Johnson. JDJ Wordware, P.O. Box 354, Cupertino, CA 95015. 415-965-3245. Internet: `M.John@Sierra.Stanford.edu`.
- **JPL**

  Contact: Peter Scott. Internet: `pjs%grouch@jpl-mil.jpl.nasa.gov`. 818-354-2246.
- **Kinch Computing**

  501 South Meadow Street, Ithaca, NY 14850. 607-273-0222. FAX: 607-273-0484.
- **LaserPrint**

  P.O. Box 35, D-6101 Fränkisch Crumbach, Federal Republic of Germany. +49-6164-4044.
- **Ling**

  Contact: Fuyun Ling. 202 Chestnut Ave., Jamaica Plain, MA 02130. Internet: `lingfuyun@nuhub.acs.northeastern.edu`.
- **Long**

  Contact: Jeff Long. Internet: `jlong@blackbird.afit.af.mil`. Mr. Long is only willing to distribute minimal files to those who can't FTP and who already have the bulk of the Beebe Driver package.
- **Louisiana State University**

  Contact: Neal Stoltzfus. Department of Mathematics, Louisiana State University, 382 Lockett Hall, Baton Rouge, LA 70803. 504-388-1570.
- **Massachusetts Institute of Technology**

  Contact: Chris Lindblad. MIT AI Laboratory, Room 733, 545 Technology Square, Cambridge, MA 12138. Internet: `Cjl@Reagan.ai.Mit.edu`.
- **Max-Planck-Institut für Aeronomie**

  Contact: Helmut Kopka. Max-Planck-Institut für Aeronomie, Katlenburg-Landau, D3411, Federal Republic of Germany. 49-556-41451. Bitnet: `Mio401@Dgogwd01`.
- **n^2 Consultants**

  Contact: Norman Naugle. P. O. Box 2736, College Station, TX 77841. 409-845-3104. Internet: `Naugle@Ee.Tamu.edu`.
- **Neumann**

  Contact: Gustav Neumann. Bitnet: `Neumann@Awiwuw11`.
- **Northlake Software**

  Contact: David Kellerman. 812 SW Washington, Portland, OR 97205. 503-228-3383. UUCP: `nls!davek`.
- **Océ-Nederland**

  Contact: Jan van Knippenberg. Office Automation, P.O. Box 101, 5900 MA VENLO, The Netherlands. 0-77-592222.
- **OCLC**

  Contact: Tom Hickey. 6565 Frantz Road, Dublin, OH 43017. 616-764-6075.

- **Ohio State University**
  Contact: Ms. Marty Marlatt. Ohio State University, Department of Computer and Information Science, 2036 Neil Avenue, Columbus, OH 43210. The drivers are distributed on either ANSI or TOPS-20 DUMPER tapes, with hardcopy documentation. There is a $125 service charge (payable to Ohio State University) to cover postage, handling, photocopying, etc.
- **Oxford [1]**
  Contact: Paul Leyland.
  Janet: `pcl@robots.ox.ac.uk`.
- **Oxford [2]**
  Contact: Dr. P.S. Aspinwall. Oxford University Department of Theoretical Physics, Keble Road, Oxford, OX1 3RH, United Kingdom. 44-865-273954. Internet: `aspin@vax.oxford.ac.uk`. Janet: `aspin@uk.ac.oxford.vax`.
- **Personal TEX**
  Contact: Lance Carnes. 12 Madrona Street, Mill Valley, CA 94941. 415-388-8853. Telex: 510-601-0672.
- **Philips Kommunikations Industrie AG**
  TEKADE Fernmeldeanlagen, Attn. Dr. J. Lenzer, Thurn-und-Taxis-Str., D-8500 Nürnberg, Federal Republic Germany. +49-911-5262019.
- **Prime distribution tape**
  Contact: John M. Crawford. Computing Services Center, College of Business, Ohio State University, 1775 College Road, Columbus, OH 43210. 614-292-1741. Bitnet: `Craw4d@Ohstvma`. Internet: `Crawford-j@OSU-20.ircc.Ohio-State.edu`.
- **Procyon Informatics**
  Contact: John F. Roden. Glendenning House, 7-8 Wicklow St., Dublin 2, Ireland. 353-1-791323.
- **Quixote**
  Contact: Don Hosek. 440F Grinnell, Claremont, CA 91711. Bitnet: `DHOSEK@hmcvax`.
- **Radel**
  Contact: Jon Radel. P.O. Box 2276, Reston, VA 22090. Software is distributed on 5.25″ 360K floppy disks. For floppies sent with a return mailer, there is a charge of $1.50/floppy U.S. orders, $2/floppy elsewhere. For orders where floppies are supplied by Jon Radel, there is a charge of $5/floppy for U.S., Mexican and Canadian orders, $6/floppy elsewhere. 3.5″ 720K disks are also available for the cost of any two floppies each.
- **Radical Eye Software**
  Contact: Tom Rokicki. Box 2081, Stanford, CA 94309. 415-326-5312.
- **Research Triangle Institute**
  Contact: Randy Buckland.
  Internet: `rcb@rti.rti.org`. The program is available in the `comp.sources.misc` archives on Internet and Usenet.
- **Scan Laser**
  Contact: John Escott. 6 Churchill Close, Hartley Wintney, Nr Basingstoke, Hants RG27 2RA, England. +44-1-638-0536.
- **Science Applications**
  San Diego, CA. 619-58-2616.
- **Stanford CMS distribution tape**
- **Stanford DEC-20 distribution tape**
- **Stanford VMS distribution tape**
  Contact: Maria Code. Data Processing Services, 1371 Sydney Drive, Sunnyvale, CA 94087.
- **Stichting Acad Rechenzentrum Amsterdam**
  Contact: Han Noot. Stichting Math Centrum, Tweede Boerhaavestraat 49, 1091 AL Amsterdam, The Netherlands.
- **SullivanSFT**
  P. O. Box 292431, Lewisville, TX 75029.
- **Sun**
- **Systemhaus für Elektronisches Publizieren**
  Contact: Robert Schöninger. Arndtstrasse 12, 5000 Köln, Federal Republic of Germany.
- **Talaris**
  Contact: Sam Hassabo. 619-587-0787.
- **Technical Research Center of Finland**
  Contact: Tor Lillqvist. VTT/ATK, Lehtisaarentie 2, SF-00340 Helsinki, Finland. +358-04566132. Bitnet: `Tml@Fingate`.
- **Technische Hochschule Darmstadt**
  Contact: Klaus Guntermann. Fachbereich Informatik, Insitut für Theoretische Informatik, Alexanderstrasse 24, D-6100 Darmstadt, Federal Republic of Germany. Bitnet: `XITIKGUN@DDATHD21`.
- **Technische Universität Braunschweig**
  Contact: Georg Bayer. Bitnet: `C0030001@Dbstu1`.
- **TEX Users Group**
  P.O. Box 9506, Providence, RI 02940-9506. 401-751-7760. Internet: `tug@math.ams.com`.
- **Texas A&M [1]**
  Contact: Bart Childs. Department of Computer Science, Texas A&M University, College Station, TX 77843-3112. 409-845-5470. Internet: `childs@cs.tamu.edu`.

- **Texas A&M [2]**
  Contact: Ken Marsh. Thermodynamics Research Center, Texas A&M University, College Station, TX 77843. 409-845-4995. Bitnet: `KMarsh@Tamnil`.

- **Texas A&M [3]**
  Contact: Thomas Reid. Computing Services Center, Texas A&M University, College Station, TX 77843. 409-845-8459. Bitnet: `X066TR@TAMVM1`.

- **TEXsys**
  Contact: Joachim Schrod. Kranichweg 1, D-6074 Rödermark, Federal Republic of Germany. +49-6074-1617.

- **Tools GmbH Bonn**
  Contact: Edgar Fuß. Kaiserstraße 48, 5300 Bonn, Federal Republic of Germany. UUCP: `...unido!bnu!fuss`.

- **The Toolsmith**
  P.O. Box 5000, Davis, CA 95617. 916-753-5040.

- **TPSoftware**
  Contact: Harold T. Stokes. P.O. Box 922, Provo, UT 84603-0922.

- **Università Degli Studi Milan**
  Contact: Dario Lucarella. 02/23.62.441.

- **Universität des Saarlandes**
  Contact: Prof. Dr. Reinhard Wilhelm. FB 10 Informatik, Im Stadtwald 15, D-6600 Saarbrucken, Federal Republic of Germany. UUCP: `wilhelm@sbsvax.UUCP`.

- **University of British Columbia**
  Contact: Afton Cayford. Mathematics - University of British Columbia, 121-1984 Mathematics Road, Vancouver V6T 1Y4, British Columbia, Canada. 604-228-3045.

- **University of California, Berkeley**
  Contact: Michael Harrison. Internet: `vortex@berkeley.edu`.

- **University of California, Irvine**
  Contact: Tim Morgan. Internet: `morgan@uci.edu`.

- **University of Heidelberg**
  Contact: Joachim Lammarsch. Bitnet: `RZ92@DHDURZ1`.

- **University of Kansas**
  Contact: Edwin Bell. Bitnet: `Bell@Ukanvax`. SPAN: `Bell@Kuphsx`. Internet: `Bell%Kuphsx.Span@Star.Stanford.edu`. Department of Physics and Astronomy, University of Kansas, Lawrence, KS 66045. 913-864-3610.

- **University of Köln**
  Contact: Jochen Roderburg. Bitnet: `A0045@DkOrrzkO`. Rechenzentrum, University of Köln, D5000 Köln 41, Federal Republic of Germany. 02211-/478-5372.

- **University of Maryland**
  Contact: Chris Torek. Computer Science Department, University of Maryland, College Park, MD 20742. 301-454-7690. Internet: `Chris@Cs.Umd.edu`. The Imagen driver may be obtained via anonymous FTP from `a.cs.uiuc.edu` in the directory `pub/TeX`, file `iptex.tar.Z` or from `mimsy.umd.edu` in the directory `tex`, file `ctex`.

- **University of Sheffield**
  Contact: Ewart North. Data Processing Unit, University of Sheffield, Western Bank, Sheffield S10 2TN, England. (0742)-78555 ext. 4307.

- **University of Sydney**
  Contact: Alec Dunn. School of Electrical Engineering, University of Sydney, NSW 2006, Australia. 02-692-2014. ACSnet: `alecd@facet.ee.su.oz`. Internet: `alecd%facet.ee.su.oz@Seismo.Css.gov`.

- **University of Utah**
  Contact: Nelson H. F. Beebe. Center for Scientific Computing, Department of Mathematics, 220 South Physics, University of Utah, Salt Lake City, UT 84112. 801-581-5254. Internet: `Beebe@science.utah.edu`. All of the Beebe drivers are distributed together. They are available on 1600bpi 9-track tape in VAX/VMS BACKUP format, Unix tar format, and ANSI D-format. Send US$100 for a copy. IBM PC floppies are available from Personal TEX or Jon Radel. The programs are available for anonymous FTP from `science.utah.edu` on the Internet; information is in the file `~ftp/00readme.txt`. A VAX/VMS binary distribution is available for anonymous FTP (password guest) from `ctrsci.utah.edu`. The file `00readme.txt` in the login directory gives details. On JANET, the programs may be obtained from the directory `aston.tex::[public.texdvi210]`. On DECnet, they are available from the DECnet file repository; for more information send mail to the DECnet address `<39937::luvisetto>`. The drivers are available from Listserv on EARN to European Bitnet users. Sending the command `GET DRIVER FILELIST` (in an interactive message, or as the first line of a mail message) to `LISTSERV@DHDURZ1`. Files are obtained with the command `GET filename filetype`.

- **University of Washington**
  Contact: Elisabet Tachikawa. Northwest Computer Support Group, University of Washington, Mail

Stop DR-10, Seattle, WA 98195. 206-543-6259. Internet: `elisabet@max.u.washington.edu`.

- **University of Wisconsin**

  Contact: Ralph Stromquist. 1210 W. Dayton Street, Madison, WI 53706. 608-262-8821.

- **Unix distribution tape**

  Contact: Elisabet Tachikawa. Northwest Computer Support Group, University of Washington, Mail Stop DR-10, Seattle, WA 98195. 206-543-6259. Internet: `elisabet@max.u.washington.edu`. The Unix distribution tape may be obtained from the Northwest Computer Support Group for $100 ($110 for foreign sites). It is available either as Unix tar blocked 20, 1600 bpi, or in 1/4″ streamer cartridges for the Sun workstation. The DEC-20 program is available on request. Checks should be made payable to the University of Washington.

- **van Oostrum**

  Contact: Piet van Oostrum. Internet: `piet@cs.ruu.nl`. UUCP: `piet@ruuinfvax.UUCP`.

- **Washington State University**

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

- **Washington University**

  Contact: Stanley Sawyer. Department of Mathematics, Campus Box 1146, St. Louis, MO 63130. 314-889-6703.

- **Weizmann Insititute**

  Contact: Malka Cymbalista. Computer Center, Weizmann Institute of Science, Rehovot 76100, Israel. 08-482443. Bitnet: `Vumalki@Weizmann`.

- **Xerox**

  Contact: Margot Nelligan. Xerox Printing Systems Division, 880 Apollo Street, El Segundo, CA 90245. 213-333-6058.

- **XOrbit**

  P.O. Box 1345, D-8172 Lenggries, Federal Republic of Germany. +49-8042-8081.

- **Yale University**

  Contact: Jerry Leichter. Bitnet: `Leichter@Yalevms`. Internet: `Leichter-jerry@cs.yale.edu`. Available for anonymous FTP from `venus.ycc.yale.edu`. Log in as anonymous and do a `CD [.DVIDIS]`. That directory contains the three required files needed to run the previewer. The image *must* be transferred using BINARY mode.

Typesetters

	CDC Cyber	HP3000	IBM MVS	IBM PC	IBM VM/CMS	Siemens BS2000	Sperry 1100	UNIX	VAX VMS
Allied Linotype CRTronic									558
Allied Linotype L100, L300P				558					
Allied Linotype L202				558					558
Autologic APS-5, Micro-5				558				558	558
Compugraphic 8400		558		558				558	558
Compugraphic 8600	558			558	558		558	558	559
Compugraphic 8800				559				559	
Harris 7500								559	
Hell Digiset			559			559			

Low-Resolution Printers — Laser Xerographic, Electro-Erosion Printers

	Amdahl (MTS)	Amiga	Atari ST	CDC Cyber	Data General MV	DEC-20	HP1000	HP9000 200	IBM MVS	IBM PC	IBM VM/CMS	Prime	Siemens BS2000	Sym-bolics Lisp	UNIX	VAX VMS
Agfa P400									545	545	545		546		546	546
Canon LBP-A2, LBP-8			546			546				546					546	546
Cordata LP300										546						
DEC LN03, LN03+			546			546				546					546	546
Golden Dawn Golden Laser 100			547			547				547					547	547
HP 2680							547									
HP 2688A							547	547								
HP LaserJet, LaserJet Plus, II, IID, IIP, III, 2000		547	547			547	547			547		548			548	548
IBM 38xx, 4250, Sherpa									548	548	548				548	
Imagen	548		548		549	549				549	549			549	549	549
Kyocera F-10xx, F-20xx			549							549					549	549
Océ 6750																549
Olympia Elsa											549					
PostScript printers		550	550			550				550	550	550		550	550	551
QMS Kiss, Smartwriter		551														
QMS Lasergrafix	551	551			551				551	551	551	551	552	552	552	552
Talaris									552	552	552				552	552
Xerox 2700II, 3700, 4045				552		552					552				552	552
Xerox 8700, 8790, 9700, 9790, 4050	552								553		553				553	553

Low-Resolution Printers — Impact and Electrostatic Printers

	Acorn	Amiga	Apple Mac-intosh	Atari ST	Cadmus 9200	Data General MV	DEC-20	HP1000	HP3000	IBM MVS	IBM PC	IBM VM/ CMS	Prime	UNIX	VAX VMS
Apple ImageWriter	553	553	553	553			553				553	553	553	553	553
Benson 9424												554			
C. Itoh 8510A											554				
Citizen 120-D		554													
DEC LA75, LP100	554			554			554				554	554	554	554	554
Epson FX/MX/JX/RX	554	554		554			554	554	554		554	555	555	555	555
Epson LQ, NEC P6/P7		555		555							555				
Fujitsu				555	555										
GE 3000														555	
HP DeskJet		555		555							555				
HP InkJet		555													
MPI Sprinter	556			556			556				556	556	556	556	556
NDK Printstar												556			
Okidata	556	556		556			556				556	556	556	556	556
Printronix	556			557		557	557				557	557	557	557	557
Texas Instruments 855											557				
Toshiba	557			557		557	557				557	557	557	557	557
Varian															557
Versatec										557		557		557	

Report from the DVI Driver Standards Committee

Don Hosek

Attendees of the Texas A&M TeX Users Group Conference will doubtless be disappointed to see the lack of the full and final version of the Level 0 Driver Standard on these pages. The public exposure of the text revealed many unforeseen difficulties in some of the aspects of the standard which have detained completion of the standard unavoidably.

However, by the time that you read this the standard should be available in its final form. For those with net access, the files will be made available from `ymir.claremont.edu` for FTP and mail server access. FTP users should get the file `level0-standard-final.tex` from the directory `[anonymous.tex.dvi-standard]`. Those without FTP access should send the command

```
send [tex.dvi-standard]level0-standard-final.tex
```

to `mailserv@ymir.claremont.edu`. This file will also be available on MS-DOS floppy disks from Jon Radel (see address on 483). The file will be a self-contained LaTeX file. There also will be a file in that directory called `standard-news.txt` which will have the status of the standards development by that time.

◇ Don Hosek
Quixote
440F Grinnell
Claremont, CA 91711
`dhosek@ymir.claremont.edu`

Resources

Updates from All Over

Barbara Beeton

The Aston Archive

Some very general information: The top-level directory at Aston is TEX-ARCHIVE, username PUBLIC, password PUBLIC. It contains files named `000DIRECTORY.LIST`, `000DIRECTORY.SIZE` and `000LAST30DAYS.FILES`, which are respectively short and long form directory listings, and the names of any files that have changed in the past month. All these files are updated *every* day, at about 0130 local time. Note that these names all start with *three* zeroes, unlike those mentioned below, which have deliberately kept to a length compatible with the lowest common denominator of computers, namely IBM.

In addition, *every* directory in the tree contains a `00FILES.TXT` which gives a reverse chronological list of all the files in the current directory. These files are only updated whenever anything changes; the programs and batch job to perform these daily updates was written by Niel Kempson. Note that the action of changing one of these files results in that for the level above noticing that the directory itself has changed, so the date of a `00FILES.TXT` in any of the first level directories TEX-ARCHIVE.* will reveal when anything in that branch of the tree was last added/updated.

A small request to TeXserver

```
<TeXserver@uk.ac.aston.tex>
```

or (in the U.S.)

```
<TeXserver%uk.ac.aston.tex
                    @nsfnet-relay.ac.uk>
```

as follows could collect all such first-level listings in one go: they'll all arrive separately, unless you care to append the `/DCLAR` qualifier to the files command, to have it send you a batch job that will reconstitute all the files:

```
FILES
[TEX-ARCHIVE.*]00FILES.TXT
```

Your address will be extracted from the headers arriving at Aston. If you wish to use a different return address for the information to be mailed back to you (perhaps avoiding certain gateways which mangle ASCII during a conversion to/from EBCDIC), then you may specify the return address (including the necessary gateway) on the line preceding the TeXserver command; prefix any such alternate address with the directive `PATH`, followed by a space. Also, if you are on a system in which the case of letters in usernames is significant, you may always wish to provide a `PATH` directive, as the incoming mailer at Aston always upcases usernames (a bug of which the supplier is aware).

Other valid TeXserver commands may be given in the first non-blank line: `HELP`, `DIRECTORY`, `FILES`, `WHEREIS`, or `SEARCH`.

The above information was provided by Brian {Hamilton Kelly}. (Thanks, Brian, and I *love* your bow tie.)

OzTEX 1.3

I have received a letter from Andrew Trevorrow, in Hyderabad, India. As of the middle of August he was nearly ready to release the OzTEX implementation of TEX 3.0.

In the letter, Andrew described one change that should prove both useful and popular: "All of TEX's large arrays are dynamically allocated according to sizes appearing in a configuration file, so users can easily change *mem_max*, *font_max*, *font_mem_size*, *hash_size*, *pool_size*, etc., without having to recompile TEX."

OzTEX 1.3 should be at the usual archives by the time you read this.

Public Domain TEX on PCs

The following information was forwarded by D. Monk, of the University of Colorado, Boulder. Note that the reference for emTEX corrects an error in *TUGboat* 11, no. 2; thanks to everyone who pointed that out.

Note: The systems mentioned below can be obtained from sources other than those indicated. Some of the internet numbers and directories may change without notice. Access method for subdirectories varies; go down one directory at a time.

1. AMS-TEX. Complete, with `.mf` sources for the fonts. ftp 134.173.4.23, directory `tex/mf/ams` and, for Russian fonts, `tex/babel/russian/fonts-uwash`. (Claremont)

2. DosTEX. TEX, LATEX, driver and fonts for Epson FX. Obtainable in SimTel archives, 26.2.0.74 or `listserv@ndsuvm` or, in Europe, via Earn trickle servers. Files are `pd1:<msdos.tex>dostex1.arc` through `...dostex6.arc`.

3. EmTEX. A complete TEX with TEX, LATEX, METAFONT, many drivers and fonts (Epson FX, HPLJ, Apple LaserWriter, etc.). In Europe ftp to 129.69.1.12, directory `soft/tex/emtex`. In USA ftp to `terminator.cc.umich.edu`, directory `msdos/text-mgmt/TeX/emtex`.

4. SbTEX. TEX only. Obtainable in SimTel archives, 26.2.0.74 or `listserv@ndsuvm` or, in Europe, via Earn trickle servers. File
`pd1:<msdos.tex>sb30tex.zip`

By ordinary mail, most of the above for PCs can be obtained for mailing costs from Jon Radel, P.O. Box 2276, Reston, VA 22090, USA. Send self-addressed envelope with 45 cents postage (4 International Reply Coupons outside USA) for his latest catalog.

The LATEX help service

Max Hailperin informs us that "The LATEX-help volunteer question-answering round-robin service has moved. Although the previously published address at `sumex-aim` will continue to work for the foreseeable future, greater reliability and speed will be achieved by instead mailing to:

`LaTeX-help@cs.Stanford.edu`

In related news, I have passed on the coordinatorship to Ed Sznyter; many thanks to him for volunteering."

TEXware from the networks

An article from Peter Flynn giving an exhaustive list of network sources for public domain and shareware implementations of TEX, METAFONT, macros, and everything related, will appear in the next regular issue (*TUGboat* 12, no. 2).

Eplain

Karl Berry

I developed the `eplain` macros as part of producing the book *TEX for the Impatient*. Unlike the book, however, they are free.

`eplain` stands for "extended plain" (or "expanded", if you like). I attempted to provide macros that would be useful to most documents, as the macros in `plain` TEXare, rather than ones for high-level, "intensional", typesetting (such as a `\chapter` command).

Specifically, I wrote macros implementing these features (in no particular order):

- left-justified displays
- double column output
- producing tables of contents
- `\hrule` and `\vrule` with a different default than 0.4 pt
- producing the time of day
- listing files verbatim
- generalized footnotes
- blank and black boxes
- citations using BIBTEX, à la LATEX

Oren Patashnik took the macros implementing the last of these, citations à la LaTeX, and put them into a separate file, '`btxmac.tex`'. (He modified them a bit at the same time.) '`eplain.tex`' `\inputs` '`btxmac.tex`', naturally.

The `eplain` distribution includes a 20-odd page user manual. Besides the features above, it describes some other definitions that may be useful to people writing their own macros. The user manual is written in Texinfo format, and therefore can be translated to a form readable by GNU Emacs, as well as printed.

You can get the `eplain` distribution via anonymous ftp from the hosts

`ics.uci.edu`
`labrea.stanford.edu`

and

`ftp.cs.umb.edu`

It is available as a compressed *tar* file and, on the latter two, also as straight text files. The file '`btxmac.tex`' is also available on its own from `labrea.stanford.edu`. I encourage other archives to redistribute `eplain`. I am also willing to send it via electronic mail to people who cannot get it any other way.

◇ Karl Berry
135 Center Hill Rd.
Plymouth, MA 02360
`karl@cs.umb.edu`

New Books on TEX

Victor Eijkhout

One of the aspects of TEX that sets it apart from other text processors is the fact that there exists an ultimate reference: *The TEXbook*. As its introduction states, this book is both for people who have never used TEX before and for the experienced hackers alike. *TEX for the Impatient* makes a similar claim: Paul Abrahams, the senior author, asked himself "What kind of book would have made it easier for me to learn TEX? What kind of book would I need now, as a more experienced user, to locate commands or functions that I never learned or only half remember?" In my opinion he, and co-authors Karl Berry and Kathryn Hargreaves, have given a successful answer to the first question. My thoughts on the second question follow below.

TEX for the Impatient has a very appealing front cover: the white rabbit from *Alice in Wonderland* (the one that exclaims "Oh dear, I shall be too late!") is sitting, looking at his watch, very impatiently. The inside of the book looks good. Computer Modern is used for the text, with a surprising but very satisfactory choice of Optima bold for headings. Thirteen chapters and an index make up the approximately 360 pages of the book.

After two inevitable chapters 'Using this book' and 'Using TEX', follows an interesting third chapter: 'Examples'. Ten page-long examples with the input on the facing page give a good impression of TEX's capabilities, and give the novice a source of commands and constructs to study (and copy).

Chapter four 'Concepts' starts the reference part of the book. Instead of merging the list of concepts treated here into the table of contents, the authors decided to print it separately on the inside of the back cover. An unusual idea, but I like it. The list is some 90 terms long, and the chapter spans 55 pages. Individual concepts are therefore treated briefly but the explanations are clear and well-written, and there are many references to the subsequent chapters which treat individual commands. In this chapter I appreciated especially the fact that the authors use the anatomical analogy for TEX's workings, and refer to it repeatedly.

Although the authors suggest that novices, after having read chapters 1–3, start looking up commands and concepts as needed from the summary of commands (chapter 13), I feel that chapter 4 is really also part of the introduction to TEX. Call it a higher introduction.

The following chapters, 5–9, treat TEX commands, grouped by subject. Here too the explanations are clear, but they are less complete than I would like them. It was a wise decision not to treat each command separately, but to tackle a few commands at a time, for instance `\hss` and `\vss`, or `\unskip`, `\unkern`, and `\unpenalty`.

Chapters 10–12 are probably a good selling point for this book: let it suffice that the titles are 'Tips and techniques', 'Making sense of error messages', and 'A compendium of useful macros'. This last chapter contains an 'extended `plain` format' (see also p. 571), defining valuable macros, such as those for cross references and left-aligned display equations. Explanations of these macros limit themselves to explanations of the way to use them. A 'Capsule summary of commands' and an index complete the book.

On the whole, I find this book very clearly written, and all its information is readily accessible. However, I was perturbed by the small errors that I found. For instance, the delimiters around `\..withdelims` commands don't grow as the authors claim; they are determined by font parameters 20 and 21 of the symbol font. Also, the remarks about the depth (height) of a `\vbox` (`\vtop`) on pages 52 and 161/2 are at odds; in most cases this dimension is the depth (height) of the last (first) box or rule. On page 52 it is stated that this value is zero if the last (first) item is kern or glue, but on page 161/2 it is stated that the value is zero if the last (first) item is not a box or rule. The first statement is incomplete for the `\vtop`, because `\vtop{\write\file{...}...}` also has a zero height; the second statement is wrong for the `\vbox`, because `\vbox{...\write\file{...}}` need not have zero depth.

As I mentioned above, the question underlying this second part of the book was "What kind of book would I need now, as a more experienced user, ...". By 'experienced user' the authors apparently do not mean an aspiring TeX hacker, since this book explains the effects of commands, but little of the large scale mechanisms connecting them.

For instance, one technique in chapter 10, 'Leaving space at the top of page', is treated in a mere five lines: the reader is told that `\vskip` does not work, but that `\topglue` does. I was particularly struck by this, since I didn't know the latter command, which is a late addition to TeX version 3. Neither here, nor in the systematic reference chapters is it mentioned whether this is a macro or a primitive. That information can only be found in the command summary; it is not even in the index, as it is in *The TeXbook*.

Another example: page 86 states that "When TeX breaks a page, it discards any sequence of glue, kerns, and penalty items that follows the break". This is rather a simplification of what really happens; one might even say that this is just not true. However, it is a convenient way of looking at things, and as long as you stick to the `plain` TeX output routine you never notice the difference.

The most obvious sign that the authors do not aim at TeX hackers is of course the fact that they repeatedly refer to *The TeXbook* for the details. On page 167 they say "if you want to get adventurous you can learn all about it from pages [...] of *The TeXbook*".

In general, this book gives good factual information, and the information is very easy to find. What it lacks are the explanations, not of commands but of mechanisms.

But, since some very handy macros are given in chapter 12, this book can be useful for people wanting to understand and modify or extend existing macros. And as an introduction, it is simply a good book.

◇ Victor Eijkhout
Center for Supercomputing
Research and Development
University of Illinois
305 Talbot Laboratory
104 South Wright Street
Urbana, Illinois 61801-2932, USA
`eijkhout@csrd.uiuc.edu`

A Proto-TUG Bibliography: Installment Three

Barbara Beeton

Two installments of a TUG bibliography have appeared in previous issues. The list below continues with references to books and articles about TeX, LaTeX, WEB and related topics, or prepared using one of these tools. Thanks to the many readers who have added to the file, and especially to Nelson Beebe, whose core bibliographies have given us a model to follow in our additions and a permanent place to file the information so that it will be accessible to all electronically.

Please continue to send in your suggestions. The elements that we want to include are detailed with the last installment (*TUGboat* 11, no. 2, p. 208).

Publications about TeX

- Neenie Billawala. Metamarks: Preliminary studies for a Pandora's Box of shapes. Technical Report STAN-CS-89-1256, Stanford University Computer Science Department, May 1989.
- Francis Bourceux. *LaTeX—la perfection dans le traitement du texte.* Editions Ciaoco, Artel, Bruxelles, Belgium, 1990. ISBN 2-87085-194-4.
- Cahiers GUTenberg, 1988–. Journal of Groupe des Utilisateurs de TeX Francophones, (group of French-speaking TeX Users).
- comp.text.tex, 1989–. This is an unmoderated Usenet discussion list about TeX.

- Jacques Desarmenien. How to run TEX in French. Technical Report STAN-CS-84-1013, Stanford University, August 1984.
- Victor Eijkhout and Nico Poppelier. Wat is TEX. *TWIOscoop*, 8(2):44–48, 1990.
- Paul M. English. Using METAFONT for original font design. August 1987 (unpublished).
- K. Cleo R. Huggins. Egyptian hieroglyphs for modern printing devices. Technical Report STAN-CS-89-1251, Stanford University, June 1988.
- Donald E. Knuth. Mathematical typography. Technical Report STAN-CS-78-648, Stanford University, February 1978.
- Donald E. Knuth. The letter S. Technical Report STAN-CS-80-795, Stanford University, April 1980.
- Donald E. Knuth. The concept of a meta-font. Technical Report STAN-CS-81-886, Stanford University, October 1981.
- Donald E. Knuth. Lessons learned from METAFONT. Technical Report STAN-CS-83-978, Stanford University, August 1983.
- Donald E. Knuth. Literate programming. *The Computer Journal*, 27(2):97–111, May 1984.
- Donald E. Knuth. Digital halftones by dot diffusion. *ACM Transactions on Graphics*, 6(4):245–273, October 1987.
- Donald E. Knuth. The errors of TEX. Technical Report STAN-CS-88-1223, Stanford University Computer Science Department, September 1988.
- Donald E. Knuth. Calling all grand wizards. *TEXhax*, 89(98), November 1989.
- Donald E. Knuth. The errors of TEX. *Software—Practice and Experience*, 19(7):607–681, July 1989. This is an updated version of the Stanford CS report.
- Donald E. Knuth. Virtual fonts: More fun for Grand Wizards. *TEXhax*, 90(11 and 12), January 1990.
- Donald E. Knuth. Virtual Fonts: More Fun for Grand Wizards. *TUGboat*, 11(1):13–23, April 1990.
- Donald E. Knuth and Joe Weening. New TEX/METAFONT sources available on Stanford's master archive. *TEXhax*, 90(13), January 1990.
- Helmut Kopka. *LATEX—Eine Einführung.* Addison-Wesley, 1990. ISBN 3-89319-199-2.
- Helmut Kopka. *LATEX—Erweiterungsmöglichkeiten.* Addison-Wesley, 1990. ISBN 3-89319-287-5.
- Steen Larsen. *LATEX på dansk.* UNI•C, Danmarks EDB-Center for Forskning og Uddannelse, 1989. ISBN 87-7252-089-2.
- Michael Lesk. GRAB—inverted indexes with low storage overhead. *Computing Systems*, 1(3):207–220, 1988.
- Franklin Mark Liang. Word hy-phen-a-tion by com-put-er. Technical Report STAN-CS-83-977, Stanford University, August 1983.
- Nederlandstalige TEX gebruikersgroep, 1989–. Journal of the NTG (Dutch-speaking TEX Users Group.
- Lynn Ruggles. Letterform design systems. Technical Report STAN-CS-83-971, Stanford University, April 1983.
- Lothar Schumann. *Professioneller Buchsatz mit TEX.* R. Oldenbourg Verlag, Munich and Vienna, 1989. ISBN 3-486-21173-0.
- David R. Siegel. The Euler project at Stanford. Technical report, Stanford University, 1985.
- Richard Southall. Designing new typefaces with Metafont. Technical Report STAN-CS-85-1074, Stanford University, September 1985.
- Michael D. Spivak. *LAMS-TEX, The Synthesis.* The TEXplorators Corporation, 3701 W. Alabama, Suite 450-273, Houston, TX 77027, USA, 1990.
- Michael D. Spivak. *The Joy of TEX—A Gourmet Guide to Typesetting with the AMS-TEX macro package.* American Mathematical Society, 2nd revised edition, 1990.
- TEXEuro, 1989–. This is an unmoderated discussion list for TEX with emphasis on European issues. To subscribe, send a request with the text `subscribe tex-euro` to `listserv@dhdurz1.bitnet`.
- TEXhax, 1987–. This is a TUG-supported moderated electronic mailing list. To subscribe, send a request to `texhax-request@june.cs.washington.edu`.
- TEXline, Malcolm Clark, editor, 1987–. This is an informal newsletter of the TEX community.
- TEXmag, 1988–. This is an electronic magazine with articles about TEX. To subscribe, send a request with the text `subscribe texmag-l` to `listserv@vm.byu.edu`.
- UKTeX, 1987–. This is an electronic discussion list for TEX issues in the United Kingdom.

Publications prepared with TEX

- Harold Abelson and Gerald Jay Sussman, with Julie Sussman. *Structure and interpretation of computer programs.* MIT Press, Cambridge, MA, 1985. ISBN 0-0262-01077-1.
- The American Bibliography of Slavic and East European Studies, Barbara Dash, editor. Library of

Congress, 1988. LaTeX for text; plain TeX for 3-column index.

- American Mathematical Society, 198x. By the end of the 1980s, almost all AMS journals and monographs have been set using TeX. This entry needs to be replaced by an explicit list of them.
- S. Angus, B. Armstrong, and K. M. de Reuck. *Chlorine Tentative Tables.* IUPAC chemical data series. Pergamon Press, Oxford, 1985. ISBN 0-08-030713-2. Typeset using TeX80.
- Richard H. Battin. *An Introduction to the Mathematics and Methods of Astrodynamics.* AIAA Education Series. American Institute of Aeronautics and Astronautics, New York, 1987. ISBN 0-930403-25-8.
- Rodney A. Brooks. *Programming in Common Lisp.* Wiley, 1985.
- Elizabeth A. Cashdan, editor. *Risk and Uncertainty in Tribal and Peasant Economies.* Westview, San Francisco, 1990.
- Hal Caswell. *Matrix Population Models.* Sinauer Associates, Sunderland, Massachusetts, 1989. ISBN 0-87893-094-9 (cloth) 0-87893-093-0 (paper).
- Malcolm Clark, editor. *Coastal research: UK perspectives,* Norwich, UK, 1984. Geo Books. ISBN 0-86094-166-3. Typeset using TeX80.
- Malcolm Clark. *pc-Portable Fortran.* Computers and their applications. Ellis Horwood, 1986. ISBN 0-7458-0005-X. Laser-printed using TeX.
- Computational linguistics, James S. Allen, editor, 1991. To be published quarterly by MIT Press; fonts not yet finalized.
- Complex Systems, Stephen Wolfram, editor. Complex Systems Publications Inc., 1987. Technical journal published six times per year, using Computer Modern fonts.
- Guy L. Curry, Bryan L. Deuermeyer, and Richard M. Feldman. *Discrete Simulation: Fundamentals and Microcomputer Support.* Holden Day, 1989. ISBN 0-8162-2080-3.
- Dam Engineering. Reed Enterprise, Reed Business Publishing Group, Room 922, Quadrant House, The Quadrant, Sutton, Surrey SM2 5AS, UK, January 1990. Publication of International Water Power & Dam Construction. Laser-printed using LaTeX.
- Arthur de Gobineau. *"Mademoiselle Irnois" and Other Stories".* University of California Press, Berkeley, 1988. ISBN 0-520-05946-8. translated and edited by Annette Smith and David Smith.
- Carl L. DeVito. *Functional Analysis and Linear Operator Theory.* Addison-Wesley, 1990. ISBN 0-201-11941-2.
- R. Kent Dybvig. *The SCHEME Programming Language.* Prentice-Hall, 1987.
- Paul J. Ellis and Y. C. Tang, editors. *Trends in Theoretical Physics.* Addison-Wesley, 1990. ISBN 0-201-50393-X.
- Electronic Publishing—Origination, Dissemination, and Design. Wiley, 1988–. This journal accepts papers in TeX form.
- Stephen A. Fulling. *Aspects of Quantum Field Theory in Curved Space-Time.* London Mathematical Society Student Texts, 17. Cambridge University Press, Cambridge, 1989. ISBN 0-521-34400-X (hardcover), 0-521-37768-4 (paperback). This book was prepared with PCTeX; the figures were prepared with PiCTeX.
- Rosalind S. Gibson. *Principles of Nutritional Assessment.* Oxford University Press, 1990. ISBN 0-19-505838-0. Set by LaTeX, with graphics from Harvard Graphics and Adobe Illustrator; designed by Ian L. Gibson and Philip Taylor with the assistance of the publisher. The book is typeset in Adobe Times Roman 10.5/12 and 8.5/10.5, with figure annotation in Adobe Helvetica. The reasons for the unusual font size are described in Philip Taylor's TeX90 conference paper, to appear in TUGboat **12**, no. 1, 1991.
- Philip E. Gill, Walter Murray, and Margaret H. Wright. *Practical Optimization.* Academic Press, London, 1981. ISBN 0-12-283952-8.
- Daniel H. Green and Donald E. Knuth. *Mathematics for the Analysis of Algorithms.* Birkhäuser, second edition, 1982.
- W. Daniel Hillis. *The Connection Machine.* MIT Press, Cambridge, Mass., 1985 (1989 softbound). ISBN 0-262-08157-1 (hardcover), 0-262-58097-7 (softbound). This book was prepared with LaTeX.
- IEEE Transactions on Electrical Insulation, 1989–.
- Eeva Ilola and Arto Mustajoki. *Report on Russian Morphology As It Appears in Zaliznyak's Grammatical Dictionary.* Number 7 in Slavica Helsingiensia. Helsinki, 1989. ISBN 951-45-4904-X. Typeset using AM and MCYR fonts.
- The Journal of C Language Translation, Rex Jaeschke, editor. 1810 Michael Faraday Drive, Suite 101, Reston, VA 22090, USA. Tel: (703) 860-0091. E-mail: `uunet!aussie!jct`, 1989–.
- Journal of Geophysical Research. The American Geophysical Union, 2000 Florida Avenue, NW, Washington, DC 20009. This journal accepts papers in TeX form.
- Samuel N. Kamin. *Programming Languages: An Interpreter-Based Approach.* Addison-Wesley, 1990. ISBN 0-201-06824-9.

- James P. Keener. *Principles of Applied Mathematics*. Addison-Wesley, 1988. ISBN 0-201-15674-1.
- Bjørn Kirkerud. *Object-Oriented Programming with Simula*. Addison-Wesley, 1989. ISBN 0-201-17574-6. Typeset using LaTeX.
- David M. Kreps. *A Course in Microeconomic Theory*. Princeton University Press, 1990. ISBN 0-691-04264-0. Typeset using PostScript and Computer Modern fonts.
- Rolfe A. Leary. *Interaction Theory in Forest Ecology and Management*. Martinus Nijhoff/Dr W. Junk Publishers, 1985. ISBN 90-247-3220-4.
- Jack London. *John Barleycorn: Alcoholic Memoirs*. Oxford University Press, Oxford, 1989. ISBN 0-19-281804-X. edited with an introduction by John Sutherland.
- Marc Mangel and Colin W. Clark. *Dynamic Modeling in Behavioral Ecology*. Princeton University Press, Princeton, NJ, 1988.
- Mathematica Journal, Stephen Wolfram, editor. Addison-Wesley, 1990.
- Neural Computation, Terrence Sejnowski, editor. MIT Press, 1989. Technical journal published four times per year, using PostScript and Computer Modern fonts.
- Yoh-Han Pao. *Adaptive Pattern Recognition and Neural Networks*. Addison-Wesley, 1989. ISBN 0-201-12584-6.
- Sebastian P. Q. Rahtz, editor. *Information Technology in the Humanities*, Computers and their applications. Ellis Horwood, 1987. ISBN 0-7458-0148-X. Laser-printed using LaTeX.
- Tom Richards. *Clausal Form Logic*. Addison-Wesley, 1989. ISBN 0-201-12920-5. Typeset using TeX.
- David F. Rogers. *Procedural Elements for Computer Graphics*. McGraw-Hill Publishing Company, 1985.
- Robert Sedgewick. *Algorithms in C*. Addison-Wesley, Reading, 1990. ISBN 0-201-51425-7.
- John Sutherland. *The Longman Companion to Victorian Fiction*. Longman, Burnt Mill, Harlow, Essex, England, 1989. ISBN 0-582-49040-5. Published in the U.S. as *The Stanford Companion to Victorian Fiction*, (Stanford: Stanford University Press, 1989).
- John Sutherland. *Mrs Humphry Ward: Eminent Victorian, Pre-eminent Edwardian*. Oxford University Press, Oxford, 1990. ISBN 0-19-818587-1.
- Tommaso Toffoli and Norman Margolus. *Cellular Automata Machines*. MIT Press, Cambridge, MA, 1987. ISBN 0-262-20060-0.
- Ib Troen and Erik Lundtang Petersen. *El Atlas Eólico Europeo*. Risø National Laboratory, Roskilde, Denmark, 1990. ISBN 87-550-1638-3. Typeset on a Canon Series III 300-dpi laser printer, with extensive graphics and data tables; the data files are available on IBM PC diskettes.
- TV Guide magazine, 1987. Portions of this magazine (20 million issues weekly) are set with TeX.
- Robert Ulichney. *Digital Halftoning*. MIT Press, 1987. ISBN 0-262-21009-6.
- VAX VMS version 4.x and 5.x manuals. Digital Equipment Corporation, 1988. The complete VMS manuals sets are produced with TeX, but authors actually prepare input in a form suitable for an earlier in-house system, which is then automatically translated to TeX form.
- Bruce S. Weir. *Genetic Data Analysis*. Sinauer, Sunderland, Mass., 1990.
- Herbert S. Wilf. *Algorithms and Complexity*. Prentice-Hall, 1986.

The 1990 DECUS TeX Collection

Ted Nieland

The DECUS Languages and Tools SIG Public Domain Working Group and the Electronic Publishing SIG TeX/LaTeX/WEB Working Group are proud to announce the 1990 DECUS TeX Collection. This collection offers nearly everything a TeX User would want on their system for TeX.

The master tapes for the collection have been sent to DECUS Library and to the top of the DECUS LUG distribution tree. The new collection will be available to all shortly through their channel for procuring DECUS Software.

This collection is an extensive rework of the previous collection with nearly all of the material being updated or new. More DVI drivers have been added and many of the VMS programs now sport a CLD interface.

Also, an extensive effort on documentation has taken place resulting in a DECUS TeX Help Library.

The following items are included in the DECUS TeX Collection 1990:

- WEB (Tangle 4.0 / Weave 4.1)
- TeX Version 3.0

- METAFONT Version 2.0
- LaTeX Macro Package 2.09 (7 Dec 1989) (with mod for TeX 3.0)
- SLITeX Macro Package 2.09 (7 Dec 1989) (with mod for TeX 3.0)
- BIBTeX Version 0.99c
- TeXsis Macro Package Version 2.13
- DVIOUT Version 1.2
- DVIPS for VMS, Version 5.35
- DVItoVDU Version 3.2
- DVItoLN03 Version 3.1-4
- XDVI (with support for DecWindows)
- TeXx (with support for DecWindows)
- Vassar Spell Version 2.2
- FWEB (including support for VMS)
- CWEB (including support for VMS)
- MWEB
- TIB
- CRUDETYPE
- DVIDIS (for VAXstations Running VWS)
- GPLOT 4.23
- RNOTOTEX
- IDXTeX
- GloTeX
- DVIDVI
- MAKEINDEX
- PICTeX
- TeXTYL
- DVI2TTY
- LSE Templates for LaTeX and BIBTeX
- MFWARE (GFtoPK, GFtoPX, etc)
- PICMODE
- TR2TEX
- WS2LATEX
- AMS-TeX Macro Package
- AMS-LaTeX Macro Package
- PHYZXX Macro Package
- PHYSE Macro Package
- ScriptTeX Macro Package
- MuTeX Package (including METAFONT files)
- Clarkson LaTeX & BIBTeX Style Collections
- DECUS TeX Help Library
- Beebe Utah DVI Driver Collection with additional submissions
- DVI2PS
- Many Font Additions (Concrete, Dürer, Chess, DECUSLOGO, among others)
- Support for foreign languages including Dutch, French, German, Greek, Hebrew, Icelandic, Italian, Japanese, Korean, Portuguese, Russian, Spanish, Thai, Turkish, and Vietnamese
- TeX for the Amiga with some DVI Drivers and the LaTeX Picture Editor (LPE)
- TeX for the Macintosh (OzTeX), along with BIBTeX, and DVI drivers
- TeX for MS-DOS, plus previewers and DVI drivers
- Various TeXware for UNIX, including WEB2C and XTeX (for DECstations)

The following output devices are supported:

- DEC LN03 (requires a RAM Cartridge) [DVITOLN03]
- DEC LN03 Plus (uses bitmaps) [DVIL3P]
- DEC LA75 [DVI175]
- PostScript (LPS40, Apple LaserWriter, LN03S) [DVIALW, DVIPS, DVIOUT, GTEX]
- Hewlett Packard Laserjet [DVIJET]
- Hewlett Packard Laserjet Plus [DVIJEP]
- Cannon Engine Laserprinter [DVICAN]
- EPSON Printer [DVIEPS]
- Printronix Printer [DVIPRX]
- Okidata Pacemark 2410 (72 or 144 DPI) [DVIOKI]
- VT terminals, ReGIS Terminals, Tektronix Terminals [DVITOVDU]
- VAXstations running VWS [DVIDIS]
- DECWindows [XDVI, TeXX]
- Version 3.10 BBN BitGraph Terminal [DVIBIT]
- Golden Dawn Golden Laser 100 printer [DVIGD]
- Imagen imPRESS-language laser printer family [DVIIMP]
- Apple Imagewriter 72 or 144 dpi printers [DVIM72 or DVIMAC]
- MPI Sprinter 72 dpi printer [DVIMPI]
- Toshiba P-1351 180 dpi printer [DVITOS]
- Generic Output [DVI2TTY]
- QMS Laser Printers [GTEX]

The collection includes numerous example files including *A Gentle Introduction to TeX* by Michael Doob and *Essential LaTeX* by Jon Warbrick.

For more information on getting a copy of the DECUS TeX Collection, contact your DECUS Local User Group or the DECUS Library at:

DECUS Library (BP02)
219 Boston Post Road
Marlboro, MA 01752-1850
(508) 480-3418/3659/3446

The IVRITEX Mailing List

Don Hosek

IVRITEX is a discussion list primarily for those using TEX to typeset Hebrew. The forum, like TEXhax, is intended for both users and implementors. The primary focus is on the use of the TEX-XET program for typesetting Hebrew text and the font problem, but discussion of issues regarding other TEX add-ons for handling R-L text and/or issues regarding other R-L languages (e.g., Arabic, Aramaic, etc.) is encouraged. An informational message summarizing the state of Hebrew TEX is posted to the list on a bi-weekly basis.

Users may subscribe to the list by sending the following command as the first line of a mail message to the Bitnet address LISTSERV@TAUNIVM:

SUBS IVRITEX *your full name*

(Your name is *not* your net-address.) Non-Bitnet subscribers may need to explicitly route all messages to TAUNIVM through an appropriate mail gateway, e.g.,

LISTSERV%TAUNIVM.BITNET@CUNYVM.CUNY.EDU

If the attempt is successful, you will receive a mail message from the listserver notifying you that you are added to the list.

There is also an archive of files for Hebrew TEX available from LISTSERV@TAUNIVM. To get a list of files available from the listserver, send the command INDEX IVRITEX to LISTSERV@TAUNIVM. For help on accessing files from the listserver, send the command HELP.

◇ Don Hosek
440F Grinnell
Claremont, CA 91711
dhosek@ymir.claremont.edu

VM/CMS Site Report

Joachim Lammarsch

The new changes for TEX 3.1 et al. are finished and will be available on the next distribution tape which can be ordered from Maria Code next January. Peter Breitenlohner, who has written all the new changefiles, has developed a new program named DVICOPY which reads a dvi-file containing virtual fonts and writes a dvi-file containing real fonts.

To make the changefiles and the new program available, I have created a new filelist at LISTSERV@DHDURZ1.BITNET named VM-CMS which contains the new files.

To get an index send the command

```
GET VM-CMS FILELIST
```

to the server. To get a file send the command

```
GET filename filetype VM-CMS
```

Further news will be announced in the discussion list TEX-IBM. To subscribe to this list send the command

```
SUB TEX-IBM firstname familyname
```

to your nearest listserv.

Dean Guenther has made available the whole distribution tape via FTP from

WSUVM1.CSC.WSU.EDU

(134.121.1.39). Login with the account TEX, password is GUEST.

◇ Joachim Lammarsch
Computing Center
University of Heidelberg
Im Neuenheimer Feld 293
6900 Heidelberg 1
Germany
Bitnet: X92@DHDURZ1

Typesettings on PCs

TEX Implementations for IBM PCs: Comparative Timings

Erich Neuwirth

Timing tests were performed on several implementations of TEX for IBM PCs (and compatibles). These were the tested versions:

PCTEX	2.93 (2.9b)
PubliCTEX	2.99
SBTEX	2.9
SBTEX	3.0
μTEX	2.96.2
DOSTEX	2.93a
emTEX	2.99 [2g]
emTEX	3.0 [2h]

The files used for testing:

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

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

- Text6 is Barbara Beeton's review and the Boston Computer Society mathematical text processor benchmark from *TUGboat* 6, no. 3. It (naturally) contains complicated formulas and uses the *TUGboat* style. The document is 4 pages long.

Table 1 shows the times associated with the tests.

The following special events occurred during the benchmark:

(1) hash table size exceeded, program stopped.

(2) capacity exceeded, program stopped.

(3) Text1, Text4, Text5, and Text6 under PCTEX had to be run with additional command line parameters, `/f=26000 /m=65000` in our case.

PubliCTEX (in normal run) used the hard disk for virtual memory

If we take SBTEX 3.0 as the base for a comparison of performance, we get relative indices of performance as shown in Table 2 (a low value indicates fast performance).

All programs were run on a 25MHz 386 machine with MSDOS 3.3 installed. All programs were run with QEMM-386 installed giving 960K of simulated expanded memory and 600KB free main memory.

Table 1. Test times

	Text1	Text2	Text3	Text4	Text5	Text6
PCTEX	12:11 (3)	1:28	3:39	1:07 (3)	2:15 (3)	1:11 (3)
PublicTEX	27:32	2:28	6:17	1:48	7:59	3:20
SBTEX	14:28	1:26	3:27	1:01	2:14	1:11
SBTEX 3.0	13:35	1:21	3:14	1:01	2:04	1:04
μTEX	15:51	(1)	(2)	1:19	3:23	1:25
emTEX	16:31	(1)	3:28	0:58	2:03	1:04
emTEX286	11:32	(1)	3:17	0:56	2:00	1:01
bigemTEX	30:51	2:21	7:18	2:01	3:32	1:55
bigemTEX286	29:38	2:17	7:10	1:59	3:31	1:55
emTEX 3.0	15:05	1:14	3:11	0:56	1:48	1:03
emTEX286 3.0	14:44	1:12	3:02	0:51	1:46	0:59

Table 2. Relative performance

	Text1	Text2	Text3	Text4	Text5	Text6
PCTEX	0.90	1.09	1.13	1.10	1.09	1.11
PublicTEX	2.03	1.83	1.94	1.77	3.86	3.12
SBTEX	1.07	1.06	1.07	1.00	1.08	1.11
SBTEX 3.0	1.00	1.00	1.00	1.00	1.00	1.00
μTEX	1.17			1.30	1.64	1.33
emTEX	1.22		1.07	0.95	0.99	1.00
emTEX286	0.85		1.02	0.92	0.97	0.95
bigemTEX	2.27	1.74	2.26	1.98	1.71	1.80
bigemTEX286	2.18	1.69	2.22	1.95	1.70	1.80
emTEX 3.0	1.11	0.91	0.98	0.92	0.87	0.98
emTEX286 3.0	1.08	0.89	0.94	0.84	0.85	0.92

Additionally, a real "stress" test was performed with all these implementations of TEX. Text5 was "hardened": `\Lasertrue` (a switch from `TUGboat.sty`) was added (doing real two-column setting in TEX), which usually has the consequence of needing much more TEX main memory. Additionally (in the third degree) all `\eject` commands introduced to use TEX's memory cautiously were removed from Text5.

Only bigemTEX and bigemTEX286 survived this test. The running times were:

- bigemTEX 3:34
- bigemTEX286 3:33

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

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

◇ Erich Neuwirth
Institute for Statistics and Computer Science
University of Vienna
Universitätsstraße 5/9
A-1010 Vienna, Austria
Bitnet: `A4422DAB@AWIUNI11`

Tutorials

Long-winded Endnotes and Exercises with Hints or Solutions

Lincoln Durst

This is the third in a series of tutorials written for readers interested in exploring some of the subtler areas of TEX. We illustrate the concepts described by considering tools that may prove useful to authors interested in using plain TEX during creation of manuscripts of books or articles. The first and second installments appeared in *TUGboat* 10, no. 3, pp. 390–394, and volume 11, no. 1, pp. 62–68.

In this episode, we consider discursive endnotes and exercises for which hints or solutions are given. We are interested in endnotes and exercises that may be longer than one paragraph and that may cite one another. (We treat the endnote case as a special case of the exercise case.) As in the earlier pieces in this series, one of the main ideas is to get TEX to attend to the numbering of items and to provide marginal notes in working drafts in order to facilitate cross references. Some of the central ideas here have been introduced in the earlier pieces (for example, using TEX to write things into files), but this time we will be forced to devote special attention to some subleties not considered in the earlier cases.

Source code for *The TEXbook* (part of the general TEX distribution; it is described in Appendix E of *The TEXbook*, pages 412–425) reveals that a central idea of Knuth's about exercises and their solutions is that these should not be separated from one another in the text file (*ibid.*, page 422), no matter how far apart they will eventually appear in the finished book (or books, if solutions or answers are to appear in separate units — for example, in a supplementary volume for teachers).

If we keep the exercises and their solutions together in the source file, it will be possible to rearrange their order, add more, or delete some, and let TEX do the renumbering without tampering with the coordination between the exercises and the solutions. Think of this as analogous to the way footnotes are treated in TEX: the text of each footnote is imbedded in the source file at the point at which it is cited. This is also the way complicated endnotes ought to be handled, for exactly the same reasons. And, while we're at it, there's no difficulty in arranging for solutions and exercises to appear next to each other even in draft versions of the printed text until the dust settles and preparation of pages in final form is ready to begin. In addition, some authors may find it helpful to see the text of endnotes in draft versions of the printed text at the places they are called. Code that will make this possible is described below.

In some books, the fifth edition of Harold Davenport's *The higher arithmetic* (Cambridge University Press, 1982) is one, separate sections or appendixes are provided to contain hints for the exercises and their answers. No problem.

Exercises and endnotes, some preliminaries. Eventually we will consider definitions for macros such as

1. Powers, primes, polynomials, and polygons. \sect.pppp.

Exercise 1.1. Show that if n has an odd factor greater than 1, 2^n+1 is not a prime number and, therefore, the only primes of this form are Fermat numbers. \exer.oddfact.

Consider a regular polygon of n sides with one vertex at $(1,0)$ and the others lying on the circle with radius 1 whose center is at the origin $(0,0)$. If ϕ is the angle between radii from the origin to consecutive vertices, we may write $\epsilon_n^k = \cos k\phi + i\sin k\phi$ for the complex numbers representing the vertices of the polygon; here $0 \le k \le n-1$. These n complex numbers are roots of the polynomial $x^n - 1$, which always has at least the two factors $x-1$ and $x^{n-1}+\cdots+x^2+x+1$ (sum of a geometric series, again!). If n is a prime, it can be proved (see FIGURE 2 for a pair of sources) that the second factor is irreducible. More generally, suppose that ϵ_n is a root of an irreducible polynomial of degree d. It can be proved (same sources) that if p is a prime whose square does not divide n, then d is divisible by $p-1$; and if p is a prime whose square divides n, then d is divisible by $p(p-1)$.

Exercise 1.2. Determine all values of n for which d, the degree of the irreducible polynomial with root ϵ_n, is a power of 2. \exer.power2.

Hints for exercises.

1.1. Consider the sum of the geometric series $1-x+x^2-x^3+\cdots+x^{2k}$. \exer.oddfact.

Answers for exercises.

1.1. First $1-x+x^2-x^3+\cdots+x^{2k} = (x^{2k+1}+1)/(x+1)$, so with $n=lm$, $l=2k+1$, and $x=2^m$, we have $2^n+1=(2^m+1)(1-2^m+2^{2m}-2^{3m}+\cdots+2^{(l-1)m})$. If n has no odd factors greater than 1, it must be a power of 2. \exer.oddfact.

1.2. Suppose p is any prime dividing n. If p^2 divides n, then d has the factor $p(p-1)$ which must be a power of 2, so p is 2; and if p^2 does not divide n, then $p-1$ still divides d which means that $p-1$ is a power of 2. Therefore (cf. Exercise 1.1) all such n have the form \exer.power2.

$$n = 2^m F_{n_1}\cdots F_{n_r},$$

where $m \ge 0$ and all the factors after 2^m, if any, are distinct Fermat primes, F_i (of which only five are known, $0 \le i \le 4$).

FIGURE 1

```
\exercise #1\exer #2\hint #3\answ #4\endexer
\endnote #1\note #2\endendn
```

that will do all the things we want. Parameter #1 provides a "tag" to appear in the name of the macro that will be used to cite the exercise or note (as in the previous tutorial); parameters #2, #3, and #4 represent the texts of the exercise or the note, the hint, and the solution, respectively. While working with the material on the screen, you may find it convenient to begin a new line (indented) for each of the delimiting control sequences, say,

```
   \exercise ⟨tag⟩
   \exer ⟨text of exercise⟩
⟨more of the same⟩
   \hint ⟨text of hint⟩
⟨more hint⟩
   \answ ⟨text of solution⟩
⟨solution, solution, solution⟩
   \endexer
```

or

```
   \endnote ⟨tag⟩
   \note ⟨text of endnote⟩
⟨more of the same⟩
   \endendn
```

This should make it easier to find your way around during revision than it might be if everything were rolled up together into one big wad.

If you're not going to distinguish between hints and solutions or if you're never going to cite any exercises, you can get away with fewer than four parameters, of course. In our examples for exercises we include all four and leave for the reader omission of any considered superfluous for a given project. In order to provide for the possibility that the text of the endnote or the exercise itself, the hints, or the solutions, may run to more than one paragraph, we use `\long\def` s. For the preprocessing run (see the previous tutorial in this series), we can set (in `prepare.tex`):

```
\long\def
\exercise #1\exer #2\hint #3\answ #4\endexer
   {\Exer{#1}}
\long\def
\endnote #1\note #2\endendn{\Endn{#1}}
```

(Notice that, with these definitions, the texts of exercises, hints, solutions, and endnotes will be discarded during preprocessing. Only the tags for the macro names are of interest at that stage.) In addition to the definitions above, we put the following code in `prepare.tex`, mimicking definitions in the previous tutorial for `\Section` and `\Eqnum` (page 65, column 1):

```
\newcount\exernum \exernum=0
\def\Exer #1{\global\advance\exernum by 1
   \xdef\Exernum{\Itemnum .\the\exernum}%
   \writedef[Exer//#1//Exernum]}
\def\exer.#1.{\csname Exer#1\endcsname}
\newcount\endnnum \endnnum=0
\def\Endn #1{\global\advance\endnnum by 1
   \xdef\Endnnum{\the\endnnum}%
   \writedef[Note//#1//Endnnum]}
\def\note.#1.{\csname Note#1\endcsname}
```

For composition runs in proof mode, hints, solutions, and endnotes could be embedded in the printed text, say on a narrower line measure with smaller type (as in FIGURE 3) in order to distinguish them more easily from the text itself; for this, include the following code in `prepare.tex`:

```
\newif\ifEmbedded \Embeddedfalse
\def\EmbedNotesEtc{\Embeddedtrue
   \ShowMacros}
```

and include `\EmbedNotesEtc` as an option in the driver file. Comment out this option when pages are being prepared for final copy, or for copy fitting in preparation for making final copy, at which time the hints, solutions, and texts of the endnotes should instead be written into the files to be used to typeset them (call these files ⟨*jobname*⟩`.hnt`, ⟨*jobname*⟩`.ans`, ⟨*jobname*⟩`.not`, say).

Endnotes and exercises: First versions. The ideas involved are the same in both cases, but `\endnote` is simpler to work with, so we consider it before `\exercise`. Here is code for `compose.tex`, beginning with the allocation of a file to hold the notes. Notice the extra counter, `\EndNs`. In the special case of endnotes, it serves no useful purpose, since it keeps step with `\endnnum`; for exercises, however, where there will be an option to skip hints or answers, the second counters will function as "pseudo-booleans": zero before the first item in question and positive otherwise. Thus they will be useful for decisions that must be made (a) to open ⟨*jobname*⟩`.not`, `.hnt`, or `.ans` whenever the first endnote, hint, or solution is encountered, and (b) to input the file in question, if it isn't empty, when its contents are to be printed.

```
   %%% Endnotes (simplest form) %%%
\newwrite\endns \newcount\EndNs \EndNs=0
\long\gdef
\endnote #1\note#2\endendn{\Endn{#1}%
   \unskip$^{\Endnnum}$\
      \begingroup\notefont
   \ifEmbedded
      \par{\narrower\parindent=0pt
      \Endnnum.\vadjNote\margtext\
         #2\strut\par}\noindent
   \else
      \global\advance\EndNs by 1
      \ifnum\EndNs=1%
         \immediate\openout
            \endns=\jobname.not
         \immediate\write\endns
            {\topline}%
      \fi
      \ifShowingMacros
         \expandafter\edef
            \csname enmn\Endnnum
            \endcsname{\margtext}%
      \fi
      \immediate\write\endns
         {\Endnnum.\ \noexpand\vadjNote
         {\csname enmn\Endnnum
            \endcsname}}%
      \Write\endns{#2\unskip\newpar}
   \fi\endgroup}  % end \notefont
```

Here the first case (`\ifEmbedded`) is the easiest: The note is printed in the text where it is cited and no writing to files is involved (see FIGURE 3). In the `\else`-case, various things occur. For the first note encountered the file is opened and something is written at the top of the file (perhaps only something like `\parindent=0pt` or `\parskip=.2\baselineskip`, etc.; in special cases — see below — there may be other things, as well). Next the macros to be used for cross references are defined and, as in the previous tutorials, there are two cases corresponding to the option of printing macro names in the margins of drafts or not printing them; in both cases the text of the note is written into the file ⟨*jobname*⟩`.not`. (Observe that, for `\ShowingMacrosfalse`, the text of the marginal note is just `\relax`.) For the present, we can define

\Write to be \immediate\write; we will describe another choice later, after we consider some of the risks involved in all this. \newpar may be simply \par to end the note, or it could also provide a \strut or \vskip to create space between adjacent notes.

The subtlest part of the code so far is the method used to send the text of the marginal notes into the file so they will be properly identified when they are read back in. Because the marginal notes are saved for later use, it must be possible to distinguish between them; this problem did not arise in the earlier tutorials because the notes were used immediately after being created. Here we identify these marginal notes by incorporating the endnote numbers in the names of the macros that expand to the texts of the notes. The macros are named \enmn\Endnnum, i.e., \enmn1, \enmn2, etc., which may look strange to any but the more critical readers of page 40 in *The TeXbook*. When using \csname, nonalphabetic characters may appear in macro names: See lines 4 and 5 in the second full paragraph on page 40. Of course, one can never actually write \enmn1 or \enmn2, but can only write \csname enmn\Endnnum\endcsname instead. As a matter of fact these macro names are never used by the author, they are written and seen only by TeX; when expanded they produce the macro names printed in the margin; recall that \margtext is defined in compose.tex as follows:

```
\gdef\margtext{%
   $\backslash$\lowercase{#1}.#2\unskip.}
```

where #1 and #2 are the arguments of \MakeNote (in this case #1 is Note and #2 is the "tag", Source, say, in FIGURE 2). For reasons given in the previous tutorial, the user writes only \note.#1. when making a cross reference to one of the endnotes, here #1 is the "tag" used to identify it. This trick with \csname is what facilitates forward references, as explained last time (page 65, foot of column 1).

\exercise will be handled analogously, the only difference being that there are more components to be juggled. Before going further, we simplify things a bit in order to reduce duplication of code by creating a generic file-writing macro, based on \endnote, for use in the other cases. Let us, therefore, rewrite the definition of \endnote as follows:

```
\long\def
\endnote #1\note#2\endendn{\Endn{#1}%
   \unskip$^{\Endnnum}$\
      \begingroup\notefont
   \ifEmbedded
      \par{\narrower\parindent=0pt
      \Endnnum.\vadjNote\margtext\
         #2\strut\par}\noindent
   \else
      \WriteFile\Endnnum\EndNs\endns %
      {not}{enmn}{#2}%
   \fi\endgroup} % end \notefont
```

where

```
\long\def
\WriteFile#1#2#3#4#5#6{\def\Number{#1}%
   \def\Counter{#2}\def\FileReg{#3}%
   \def\FileExt{#4}\def\Ident{#5}%
   \global\advance\Counter by 1
   \ifnum\Counter=1
      \immediate\openout
         \FileReg=\jobname.\FileExt
      \immediate\write\FileReg
         {\topline}%
   \fi
   \ifShowingMacros
   \expandafter\edef
      \csname\Ident\Number
         \endcsname{\margtext}%
   \fi
   \immediate\write\FileReg
      {\Number.\ \noexpand\vadjNote
         {\csname\Ident\Number
            \endcsname}}%
   \Write\FileReg{#6\unskip\newpar}}
```

With all this in hand, we can write down the definition for \exercise immediately:

```
   %%% Exercises (simplest form) %%%
\newwrite\hints \newwrite\answs
\newcount\Hints \Hints=0
\newcount\Answs \Answs=0
\long\gdef
\exercise #1\exer #2\hint #3\answ #4\endexer
   {\Endn{#1}%
   \smallskip\noindent
     {\bf Exercise \Exernum.}\
      \ifShowingMacros
         \vadjNote\margtext
      \fi
      #2\par\begingroup\answfont
   \long\def\argiii{#3}\long\def\argiv{#4}%
   \ifEmbedded
      \ifx\argiii\relax\else
         {\narrower\parindent=0pt
            Hint:\ #3\par}
      \fi
      \ifx\argiv\relax\else
         {\narrower\parindent=0pt
            Answer:\ #4\par}
      \fi
   \else
      \ifx \argiii\relax\relax\else
         \WriteFile\Exernum\Hints\hints
            {hnt}{exmn}{#3}
      \fi
      \ifx \argiv\relax\relax\else
         \WriteFile\Exernum\Answs\answs
            {ans}{exmn}{#4}
      \fi
   \fi\endgroup}  % end \answfont
```

Three classical construction problems.

We[1] propose to treat of geometrical constructions, and our object will not be so much to find the solution suited to each case as to determine the *possibility* or *impossibility* of a solution.[2]

Three problems, the object of much research in ancient times, will prove to be of special interest. They are

1. *The problem of the duplication of the cube* (also called the *Delian problem*).[3]
2. *The trisection of an arbitrary angle*.[4]
3. *The quadrature of the circle*, *i.e.*, the construction of π.[5]

In all these problems the ancients sought in vain for a solution with straight edge and compasses, and the celebrity of these problems is due chiefly to the fact that their solution seemed to demand the use of appliances of a higher order. In fact, we propose to show that a solution by the use of straight edge and compasses is impossible.

Notes.

1. The text shown here appears on page [1] of Felix Klein's little book *Famous problems of elementary geometry*, based on lectures first given by Klein in Göttingen, Easter vacation, 1894. `\note.Source.`

The English translation was made originally by W. W. Beman and D.E. Smith (Ginn & Co., 1897), and revised by R. C. Archibald (Stechert, 1930) based on the latter's article in the *American Mathematical Monthly*, volume 21, 1914, pages 247–259. *Famous problems* was later reprinted by Hafner (1950), Dover (1956), and Chelsea (1955, 1962, 1980).

2. The main result used to settle questions concerning constructions possible with straight edge and compasses is the following theorem: *If x, the quantity to be constructed, depends only upon rational expressions and square roots, it is a root of an irreducible equation $\phi(x) = 0$* [with rational coefficients], *whose degree is always a power of* 2. See page [5] of the book cited in Note 1. `\note.Solution.`

3. The irreducible polynomial in question is $x^3 - 2$, whose degree is certainly not a power of two, *loc. cit.*, page [13]. `\note.Delian.`

4. In this case, Klein works with the equation $x^3 = \cos\phi + i\sin\phi$ in the complex plane, *loc. cit.*, pages 14, 15. `\note.Trisect.`

For another approach, see Louis Weisner, *Introduction to the theory of equations*, Macmillan, New York, 1938, pages 159–162. Weisner bases his argument on the irreducibility, over the field of rational functions $R(t)$, of the equation $4x^3 - 3x - t = 0$ satisfied by $t = \cos\theta$, $x = \cos\frac{\theta}{3}$.

5. In 1882, Lindemann proved [*Mathematische Annalen*, volume 20] that π is a transcendental number, *i.e.*, it is not the root of any polynomial with rational coefficients and, therefore, certainly not a root of one with degree a power of 2. (Cf. Klein, *ibid.*, Part II.) `\note.Pi.`

FIGURE 2

When either a hint or an answer is to be omitted, its argument should be `\relax` or, to be on the safe side, `\relax %` (see below).

So much for the simple stuff! The code listed above can actually be used to produce the results shown in the FIGURES 1–3 (defining `\Write` to be `\immediate\write`), but would fail for examples somewhat more complicated. The rest of this discussion is concerned with limitations of the procedure described above and with ways those limitations may be circumvented.

Files read by TeX. In the first place, when TeX writes something into a file, it writes one line at a time, so that the kind of thing we have been discussing can result in some very long lines: Whole paragraphs, or collections of several paragraphs, if they appear as a single argument, will be strung out into gigantic lines in the files we have been constructing. The trouble with this occurs as TeX tries to read long lines when the files are `\input` so their contents can be typeset. Some of the lines produced in making the figures for this tutorial (using the code described above) contain seven or eight hundred characters. When lines from a file being `\input` are read, TeX writes them to a buffer, which, for some implementations, can accomodate only a thousand or so characters and in some cases, even fewer (cf. *buff_size*, in *TeX: The Program*, sections 11 and 30).

For exercises, this problem may not be serious, since hints tend to be terse, and answers (and even solutions!) probably should be kept succinct on grounds unrelated to typographical questions. With endnotes, however, the situation is frequently quite different, especially in fields such as history or philosophy, where such self-restraint may be less common than in some of the sciences.

Questions to examine: (a) What factors contribute to the length of the lines written into files by TeX? (b) How can the lines be broken into smaller pieces? There is more than one answer to each of these questions.

Why are some of these lines so long? Aside from the fact that some authors can be long-winded, any macros in the lines are expanded when TeX writes them into a file. For example, the macro `\TeX` whose name has only four characters expands into more than fifty characters, as shown on page 66 of *The TeXbook*; and even something as simple as `\bf`, an umlaut, or a circumflex, can generate a dozen or so characters when expanded. Maybe there is no limit to the number of characters one could obtain by expanding such things. Thus brevity on the part of authors cannot solve the whole problem, although it surely may help to alleviate it somewhat.

Breaking the lines into shorter ones. Two variations on `\obeylines` (*The TeXbook*, pages 94, 352) can be used to cut the text into pieces corresponding to linebreaks that are present in the source file; one of these methods is simple and relatively straightforward, the other is more intricate and accomplishes more. Both methods use versions of the macros defined above; they differ in two ways, most significantly in the definitions chosen for the macro `\Write`.

As with `\obeylines`, it must be possible to recognize line-ends in the input file: This is done by converting the (ordinarily invisible) end-of-line mark, `^^M`, to an "active" character.

Two problems require special care when one attempts to diddle with TeX fundamentals as we are about to do. First, it is very important to confine such irregularity with care; hence:

```
\def\endlineactive{\begingroup
  \catcode`\^^M=\active}
```

Under normal circumstances, `^^M` is simply replaced by a space (*The TeXbook*, page 351), but the simpler of the methods here (first suggested to me by Ron Whitney) calls for making `^^M` perform the function of the TeX primitive `\newlinechar`. [See Peter Breitenlohner, *TUGboat*, volume 11, number 1, page 62.] So, we can define:

```
{\catcode`\^^M=\active
\def\breaklines{\endlineactive%
   \let^^M=\relax%
   \newlinechar=`\^^M}}%
\let\fixlines=\breaklines
```

Note that, below, the pair `\fixlines`, `\endgroup` delimits the region in which `^^M` will be permitted its strange behavior.

The other problem that requires special attention is that it is impossible to monkey with TeX's system of categories after the text involved has been seen by TeX. Therefore we shall cut the macros for endnotes and exercises into two pieces, so we can make the switch after TeX sees the tags and the text of the exercise but before TeX sees the material to be put into the files: the text of the notes, hints, and solutions. We therefore define, in `compose.tex`,

```
\gdef\Endnote #1\note{\def\tag{#1}%
   \fixlines\Note}
\long\def\Note #1\endendn{\endnote\tag %
   \note #1\endendn\endgroup}%end \fixlines

\long\gdef
\Exercise #1\exer #2\hint{\def\tag{#1}
   \long\def\xtext{#2}\fixlines\Hint}
\long\def
\Hint #1\answ #2\endexer{\exercise\tag %
   \exer\xtext\hint #1\answ #2\endexer %
   \endgroup} % end \fixlines
```

In both these cases, we simply define `\Write` to be `\immediate\write`. What we get in the files, then, is a series of lines each of which stands between a pair of end-of-line characters in the arguments containing the notes, hints, and answers. Two observations are appropriate in connection with this first method: (a) All macros will still be expanded (one of the lines produced this way in one of the figures still has over 150 characters in this case), (b) If any line contains a comment (`%...`), it does *not* contain an end-of-line character so it will be run in with its successor (with a space between them only if a real space precedes the percent sign terminating the first of the pair of lines; spaces after control sequences do not qualify, of course). Moreover, because macros are still expanded, the file may well contain an at-sign, `@`, which turns up in the expansions of quite a few so-called "private control sequences" (see *The TeXbook*, pages 344–364); hence, in this case (as well as in the original case, where no lines are broken), `\topline` should contain `\catcode'\string\@=11`, as a safety measure.

The first version (just described) is easy to explain, takes a little care in its use, and will be sufficient in many situations. The second version, to be described next, is considerably trickier (the tricks are all in the definition of `\Write`), and it does much more. In particular, because it does not expand macros, it not only produces files whose lines are even shorter, but the results are more easily read by ordinary humans. It is also useful in a number of situations other than the one that concerns us here. The central idea is described in an article by Ron Whitney in this issue of *TUGboat* ("Sanitizing control sequences under `\write`", p. 620). Next we apply simplified versions of some of the ideas in that article to our problem (simplified because, for endnotes and answers, we need not worry about the number of the page on which the note is cited or the exercise is stated). When one makes a table of contents, for example, a more complicated version will be required, as discussed in the article just mentioned.

Breaking lines without expanding macros. The big trick described in Ron Whitney's report involves using the TeX primitive `\meaning`. `\meaning` takes a token as its argument and, in the case where that token is a defined control sequence, produces a string of "characters" which appear in the argument's expansion. Thus `\meaning\cos` expands to

```
macro:->\mathop{\rm cos}\nolimits
```

What follows the "arrow" here (`:->`) *looks like* the expansion of `\cos`, but—as explained in the sanitizing article—it is not *really* the expansion because all the characters shown above (except for the space) have category 12 (other: none of the above or below) instead of what they ought to have (backslash should really have category 0, left and right curly braces categories 1 and 2, and the letters category 11; see *The TeXbook*, page 37).

The trick is to create a new macro whose (first-level) expansion is what you want to write into the file, apply `\meaning` to the new one, then catch the sequence following the arrow in `\meaning`'s expansion, and write *that* (which will not be expanded) into the file as a string of individual characters. Later, when the file is read, TeX won't remember how those characters got there, it will just take them for what they appear to be. Here's one way to pull this off: First define

```
\def\getMeaning#1:->#2\endget{%
   \def\Meaning{#2}}
```

`\getMeaning` may be used as follows: Suppose that `\Line` represents one of the lines in a paragraph to be sent to a file. Then consider

```
\expandafter\getMeaning\meaning\Line\endget
```

Here `\expandafter` causes `\meaning\Line` to expand to

`macro:->`⟨*text of the line*⟩

and `\getMeaning` throws out everything here not after the "arrow". As mentioned above, our code is a bit simpler than Ron's because we can ignore the number of the page on which the exercise appears or the endnote is cited.

So far we know how to avoid expansion of macros; now we have to cut the lines into shorter pieces so we can use this trick. Again we make `^^M` active in order to find the end-of-lines in the file, but this time we use the end-of-lines as terminators of strings of characters whose "meanings" will be written into the proper files one at a time. In order to distinguish the two cases, we choose for convenience

```
\def\returnactive{\begingroup
   \endlineactive}
```

For the line breaking process, we follow the method indicated in the article cited, and use a `\loop`; see *The TeXbook*, page 217, for the garden variety, and sources cited in Ron's article, for the fancy version being used here:

```
\returnactive %
\gdef\Parseline#1^^M#2\endParse{%
   \def\Firstline{#1}%
   \def\Remainder{#2}%
   }
\long\gdef\Write #1#2{%
   \let\FirstLine\empty %
   \def\Remainder{#2^^M}%
   \loop %
      \expandafter %
      \ParseLine\Remainder\endParse %
      \expandafter\getMeaning %
         \meaning\FirstLine\endget %
      \immediate\write#1{\Meaning}%
   \ifx\Remainder\empty\else\repeat %
   }endgroup % end \returnactive
```

When using this method, set

```
\let\fixlines=\returnactive
```

instead of `\breaklines`, as before.

1. Powers, primes, polynomials, and polygons. \sect.pppp.

Exercise 1.1. Show that if n has an odd factor greater than 1, 2^n+1 is not a prime number and, therefore, the only primes of this form are Fermat numbers. \exer.oddfact.

Hint: Consider the sum of the geometric series $1-x+x^2-x^3+\cdots+x^{2k}$.

Answer: First $1-x+x^2-x^3+\cdots+x^{2k}=(x^{2k+1}+1)/(x+1)$, so with $n=lm$, $l=2k+1$, and $x=2^m$, we have $2^n+1=(2^m+1)(1-2^m+2^{2m}-2^{3m}+\cdots+2^{(l-1)m})$. If n has no odd factors greater than 1, it must be a power of 2.

Consider a regular polygon of n sides with one vertex at $(1,0)$ and the others lying on the circle with radius 1 whose

Three classical construction problems.

We[1]

1. The text shown here appears on page [1] of Felix Klein's little book *Famous problems of elementary geometry*, based on lectures first given by Klein in Göttingen, Easter vacation, 1894. \note.source.

The English translation was made originally by W.W. Beman and D.E. Smith (Ginn & Co., 1897), and revised by R.C. Archibald (Stechert, 1930) based on the latter's article in the *American Mathematical Monthly*, volume 21, 1914, pages 247–259. *Famous problems* was later reprinted by Hafner (1950), Dover (1956), and Chelsea (1955, 1962, 1980).

propose to treat of geometrical constructions, and our object will not be so much to find the solution suited to each case as

FIGURE 3

Even with the fancier method there are some details to note. For example, if `^^M` is inside a group (such as `{\it ...^^M...\/}`, as in the theorem quoted in note 2, FIGURE 2), it will be passed over when the loop breaks the lines. To force the lines to break in this case, use

```
\it ...^^M...\/\rm
```

instead, say, or even

```
\bgroup\it ...^^M...\/\egroup
```

Failure to correct this problem has two consequences: you will write some long lines into the file and the `^^M`s within the groups will no longer be invisible. Thus, when you later input the file, you will lose everything after the first `^^M` between the two curly braces, up to and including the second brace and anything between that and the next `^^M`. In this situation the visible `^^M` is interpreted as a line-terminator and functions as a comment character.

In both of the cases in which `^^M` is active, it is essential to replace `\relax` by `\relax %` or the test made to determine that a hint or a solution should be omitted will fail: in these cases the argument will be `\relax ^^M` unless the end-of-line has been commented out. As Ron Whitney puts it so picturesquely, it can be difficult to induce one part of the anatomy, even TeX's, to simulate some other part and be "wholly successful" in the process.

The following code brings together the three cases we have been describing. It provides, in `compare.tex` the working definitions for `\fixlines`, `\topline`, and `\Write` for the three cases considered above.

```
\ifEmbedded
   \let\fixlines\begingroup
\else
   \gdef\newpar{\par\vskip 2pt}
   \ifexpmacros
      \let\fixlines\breaklines
      \gdef\topline{\parindent=0pt
         \catcode`\string\@=11}
      \long\gdef\Write{\immediate\write}
   \else
      \let\fixlines\returnactive
      \gdef\topline{\parindent=0pt}
      \returnactive %(*)
      \gdef\Parseline#1^^M#2\endParse{%
         \def\Firstline{#1}%
         \def\Remainder{#2}}%
      \long\gdef\Write #1#2{%
         \let\FirstLine\empty%
         \def\Remainder{#2^^M}%
```

```
            \loop %
               \expandafter %
               \ParseLine\Remainder\endParse %
               \expandafter\getMeaning %
                  \meaning\FirstLine\endget %
               \immediate\write#1{\Meaning}%
               \ifx\Remainder\empty\else%
            \repeat}%
         \endgroup % end \returnactive %(*)
      \fi %
   \fi %
```

Notice that `\returnactive` must be used not only when the lines are actually broken, but also for the definitions of `\Parseline` and `\Write` since they involve the active `^^M`. The definition to be used here for `\loop` is that in Ron Whitney's paper (note the `\else\repeat`). The switch `\ifexpmacros` is controlled by the following code in `prepare.tex` for yet another option, `\ExpandMacros`:

```
\newif\ifexpmacros \expmacrosfalse
\def\ExpandMacros{\expmacrostrue}
```

The swarm of %-signs is here to prevent the overactive `^^M`s from creating mischief. In a more advanced course, you may learn a simpler way to tame them (cf. the "sanitizing" paper in this issue).

Special challenge. Consider the endnotes in Edith Hamilton's *The Greek way* (W. W. Norton, 1942): No marks appear in the text itself, but each note indicates the page number and the line number on that page to which the note refers. Had TEX existed in 1942, how might this have been achieved?

In the next tutorial, we consider some questions related to the construction of indexes. Among other ideas, there will be more about parsing by context and some examples of other ways to use loops.

Note. A disk (5.25in DSDD) containing source text for the figures in this series of four tutorials, and the code files used to produce them, is available for MS DOS users who are members of the TEX Users Group. Send $6 (which includes a royalty for the TEX Users Group) to the address below. Outside North America, add $2 for air postage.

As usual, Ron Whitney has been generous with ideas and inspiration. He has taught me a lot about TEX — everything I know about it, except for all the things I learned from Barbara Beeton, over a period of several years, and things that I understood when I read about them for the first time in *The TEXbook*.

◇ Lincoln Durst
46 Walnut Road
Barrington, RI 02806

Output Routines: Examples and Techniques. Part III: Insertions

David Salomon

Note: Before reading this article, the reader should glance at parts I or II for disclaimers and remarks on notation.

Insertions are considered one of the most complex topics in TEX. Many users master topics such as tokens, file I/O, macros, and even OTRs before they dare tackle insertions. The reason is that insertions **are** complex, and *The TEXbook*, while covering all the relevant material, is somewhat cryptic regarding insertions, and lacks simple examples. The main discussion of insertions takes place on [115–125], where TEX's registers are also discussed. Examples of insertions are shown, mostly without explanations, on [363–364, 423–424]. There is, therefore, a need for an article like the present one. It tries to explain insertions in detail, and shows specific, simple examples. Concepts are developed gradually, and the ultimate truth revealed in steps.

Introduction

Definition: An *insertion* is a piece of a document that is generated at a certain point but should appear in the document at another point.

Common examples of insertions are footnotes, endnotes (Note 1), and floating insertions. These are important features, which explains why a general insertion mechanism has been incorporated into TEX. The following short quote (from [124]) says it all: "*This algorithm is admittedly complicated, but no simpler mechanism seems to do nearly as much.*" Using insertions, it is possible to accumulate material (text/pictures) in a box and typeset it anywhere in the document. The material can be inserted on the current page, it may be *held over* by TEX and inserted on the following page, it may be *split* between the current page and the next one, or it may wait for the end of the document. The `plain` format also provides very convenient macros, based on the general insertion mechanism, to handle footnotes and floating insertions.

A good example of insertions is the placement of index items in the right margin [423–424], an operation that is part of the *manmac* format [App. E]. See (Note 2) for an outline of the idea. A simple version is developed elsewhere in this article.

It is important to point out that, even though the insertion mechanism of TEX is general and complex, it cannot deal with every conceivable situation. Consider the case of *facing figures* (Note 3).

This is a problem that TEX's insertion mechanism cannot handle. It is easy to implement in other ways, though (Note 4).

A simple example

Before delving into the details of insertions, it is useful to develop a simple example from scratch, without using any of the built-in features for insertions. We will develop a simple mechanism for handling *floating insertions*. Suppose that diagrams should be pasted into our document (after it's been typeset) at certain points. We need to reserve room for each diagram, which is done by placing an empty `\vbox` at each insertion point.

Exercise: Why not simply say `\vskip...` or `\kern...` to reserve vertical space on the page? (Note 5).

We therefore define a macro `\Pic` by

```
\def\Pic#1 high{\par\vbox to#1{}}
```

and call it by, e.g., `\Pic 3.5in high`. The problem, of course, is that there may not be 3.5 inches of space left on the current page. In such a case, the insertion should be 'floated' to the top of the next page. We therefore have to generalize our macro such that it measures the space left on the current page before it creates the `\vbox`. To understand how this is done, the reader should first review the section on `\pagetotal` and `\pagegoal` in part I, where macro `\pagespace` was developed. This macro, whose definition is copied below, does just that.

```
\newdimen\spaceleft
\def\pagespace{%
  \ifdim\pagetotal=0pt
   \spaceleft=\vsize
  \else
   \spaceleft=\pagegoal
   \advance\spaceleft by -\pagetotal
  \fi}
```

We now generalize macro `\Pic`. It starts by setting `\box0` to the desired, empty `\vbox`. It then compares the height of the picture to the available space on the page. If there is enough room, `\box0` is simply typeset, which reserves room on the page for the diagram; otherwise, `\box0` is appended to another box, called `\fig`.

```
\newbox\fig
\def\Pic#1 high{%
  \setbox0=\vbox to #1{}
  \pagespace
  \ifdim#1>\spaceleft
   \setbox\fig=\vbox{
    \unvbox\fig\nointerlineskip\box0}
  \else
   \box0
  \fi}
```

After several calls of `\Pic`, `\box\fig` is either void, or contains a bunch of vboxes with nothing in between. When the OTR is next invoked, it first ships out the current page, then checks `\box\fig`. If that box is nonvoid, the OTR empties it by simply saying `\unvbox\fig`, which places its contents on top of the MVL, to appear at the top of the next page.

```
\output={\shipout\box255 \advancepageno
  \ifvoid\fig\else \unvbox\fig\fi}
```

This way, enough space is reserved on top of the next page for as many diagrams as necessary. It is important to say `\unvbox\fig`, rather than `\box\fig`, since this places on the MVL, not the single `\box\fig`—which is indivisible—but its contents, as separate boxes. The contents may now be spread over more than one page, if they involve many elements.

This simple example should be studied carefully, since it provides a good starting point for a full understanding of insertions.

Insertions (introductory)

On the first reading of this section, the endnotes should be ignored.

The insertion mechanism used by TEX (see [122–125]) is based on box variables. A box variable is allocated, and the `\insert` command is then used to accumulate, in that box (Note 6), vertical material to be eventually typeset (on the same page or someplace else in the document). The OTR can typeset the box anywhere on the page, using standard features, as shown below (Note 7).

Example: The command `\newinsert\fig` allocates the box variable `\box\fig`. Each command of the form `\insert\fig{`⟨*vertical material*⟩`}` accumulates material in the box (Note 8), material which TEX assumes is to be eventually typeset, by the OTR, somewhere in the document. If the material is to be typeset on the current page, TEX is instructed (see discussion of `\count\fig` below) to decrement g (Note 9) by the vertical size of the material, in order to reserve room on the page.

Just before the OTR is invoked, the insertion box becomes available (Note 10). We quote from [254] "*...just before the output routine begins, insertions are put into their own boxes.*" The OTR can typeset the material in `\box\fig` by constructions such as:

1. `\shipout\vbox{\box255\unvbox\fig}`, to typeset the insertion at the bottom of the page.
2. `\shipout\vbox{\unvbox\fig\box255}`, to typeset it at the top.
3. `\shipout\vbox{\vsplit255 to 4in \box\fig \box255}`, to typeset it 4 inches from the top of the page.
4. `\shipout\vbox{\rlap{\kern\hsize \vbox to 0pt {\box\fig \vss}}\box255}`, to place the insertion at the top right margin.

Insertions (intermediate)

The actual steps taken by TeX are more complicated. In response to the `\insert\fig` command, the material is accumulated, not in the insertion box but rather in a temporary buffer. Just before the OTR is invoked, as much of the material in the buffer as can fit on the page, is appended to the insertion box. Note that the user may, from time to time, append things to the insertion box explicitly, by means of

`\setbox\fig=\vbox{\unvbox\fig` ⟨*material*⟩`...}`

The accumulated material is eventually appended to those things. When the OTR typesets the box on the page, all the box contents go on the page; however, room on the page is reserved only for material handled through the `\insert\fig` command.

The `\newinsert` command mentioned above does more than just allocate a box. It allocates a *class of insertions*. The class includes `\count`, `\dimen`, and glue (`\skip`) variables, all of the same number, and all set to zero by default. So, for example, the `\newinsert\fig` above reserves variables `\box\fig`, `\count\fig`, `\dimen\fig`, and `\skip\fig`. They are considered class insertion `\fig`. If `\fig` happens to be 100, then the `\newinsert\fig` above allocates variables `\box100`, `\count100`, `\dimen100`, and `\skip100`.

Since `\box255` is reserved for special OTR use, only insertion classes 0...254 can be allocated. Macro `\newinsert` computes a number (counting down from 254) and allocates a box, a count, a dimen, and a skip register with that number. The reason for allocating from 254 instead of 255 is that `\box255` is reserved for special OTR use. The reason for allocating downwards is that registers `\count0`, `\count1`... are used for the page number, and that many people tend to use registers `\box0`, `\box1`... for temporary storage.

The `\dimen\fig` variable limits the size of the insertion material per page. In response to `\dimen\fig=8in` TeX will place at most 8 inches worth of insertion material from the temporary buffer in `\box\fig` per page. If the buffer contains more than 8in of material, the excess will be heldover for the next page. Placing 8 inches worth of material from the buffer in `\box\fig` may also mean that an insertion will have to be split by TeX. The splitting is done by `\vsplit` (Note 11), an operation which is also available for general use. If `\dimen\fig` is not set by the user, its value is zero, which means no room at all on the page for insertion material. The material simply accumulates in the buffer without being used, or until the value of `\dimen\fig` is changed.

The `\count\fig` variable specifies by how much g should be decremented. Setting `\count\fig=250` causes g to be decremented by 25% of the height (plus depth) of each block of insertion material placed in `\box\fig`. Example 4 above should set `\count\fig=0`, since the insertion is done on the right margin and no room should be reserved for it on the page.

The `\skip\fig` variable specifies how much vertical skip the user wants to place, by means of the OTR, on the page above or below the insertion. TeX decrements g *once* by the amount of `\skip\fig` on those pages which have some insertion material of class `\fig` in order to reserve room on the page for the skip. The skip itself, however, is not done automatically, and the OTR should not forget to add vertical glue totalling `\skip\fig` to the page.

Tracing insertions (preliminary)

A good way to understand insertions (and many other aspects of TeX) is to trace the values of the various quantities involved. Such tracing is easily done by `\message` commands, which can display many internal quantities at run time. A test of the type shown below is simple and can reveal a lot about the inner workings of insertions.

```
\hsize=3in \vsize=100pt
\output={\shipout\vbox{
 \unvbox255 \vskip\skip\fig \unvbox\fig}
 \advancepageno}

\newinsert\fig
\count\fig=1000
\dimen\fig=\vsize
\skip\fig=6pt

\message{1:t=\the\pagetotal; g=\the\pagegoal}
Text for the first paragraph
```

```
\message{2:t=\the\pagetotal; g=\the\pagegoal}
\insert\fig{⟨Material⟩}
Text for the second paragraph

\message{3:t=\the\pagetotal; g=\the\pagegoal}
\insert\fig{⟨Material⟩}
...
\bye
```

This simple experiment should be repeated with `\tracingpages=1` to get even more information on how TeX (actually, the page builder) handles insertions (see detailed examples in a later section on tracing).

Example: Endnotes

Endnotes are used in this article as a simple example of insertions. They are implemented in three steps.

1. A new class of insertions is declared and initialized by:

```
\newinsert\notes
\count\notes=0
\dimen\notes=\maxdimen
\skip\notes=0pt
```

Since the notes will be typeset on the last page, no room should be reserved for them on the current page, which is the reason for setting `\count\notes=0`. Setting `\dimen\notes=\maxdimen` guarantees that any amount of endnotes, even more than a page worth, could be placed in `\box\notes`.

2. Macro `\endnote` can be expanded anywhere in the document. It accepts one parameter, the text of the endnote, and executes `\insert\notes{#1}`. It also computes the note number, and typesets the word 'Note' and the note number in parentheses.

```
\newcount\notenumber
\notenumber=0
\long\def\endnote#1{\advance\notenumber by 1
  (Note \the\notenumber)%
  \insert\notes{\noindent[\the\notenumber]
    #1.\medskip}}
```

3. The endnotes should be typeset at the end of the document, but how? Generally, a box, such as `\box0`, is typeset by saying `\box0` or `\unvbox0`. However, we cannot do that with an insertion box, since the contents is only placed in it before the OTR is invoked. The job, therefore, has to be done in the OTR, and one way of doing it is:

```
\output={\shipout\box255
  \ifnum\outputpenalty=-20000
  \unvbox\notes\penalty-20000\fi
  \advancepageno}
```

This method uses the special penalty value of −20000, and is explained later, in the section on `\supereject`.

Each `\insert\notes` command places the material in `\box\notes` as a paragraph or as several paragraphs. Commands that apply to paragraphs in general, may be used for this material. The `\noindent` above is one example. Without the `\noindent`, the insert becomes

```
\insert\notes{[\the\notenumber]
  #1. \medskip}
```

and the material will be placed in `\box\notes` with the first paragraph indented. Another possibility is

```
\insert\notes{\narrower[\the\notenumber]
  #1. \medskip}
```

which will place the material in `\box\notes`, broken into narrow lines.

It is also possible, of course, to say

```
\insert\notes{\vbox{[\the\notenumber]
  #1. \medskip}}
```

and this will place each endnote in `\box\notes` as a `\vbox`. Such endnotes cannot be split across pages, and the last page where they appear, may come out too long or too short.

(See Lincoln Durst's article beginning on p. 577 of this issue of *TUGboat* for a different treatment of endnotes.)

Example: Footnotes

The footnotes example shown here is similar to the one implemented in the `plain` format [363], but is much simpler.

1. An insertion class `\footins` is declared and initialized by:

```
\newinsert\footins
\skip\footins=12pt plus 4pt minus4pt
\count\footins=1000
\dimen\footins=8in
```

The last line limits the amount of footnote material per page to 8 inches. If there are more footnotes than that, the excess is held over to the next page. This is automatically done by TeX's insertion mechanism. Note that preparing 8 inches worth of footnotes may necessitate splitting one footnote.

2. A `\footnote` macro is defined, with two parameters: the footnote reference symbol, and the footnote text. It typesets its first parameter and appends both parameters (without a space in between) to the insertion box.

```
\def\footnote#1#2{#1\insert\footins{
  \noindent#1#2}}
```

The footnote text may be longer than one line, but, when placed in `\box\footins`, it will be broken into lines of size `\hsize`, and will not be indented.

If the footnotes should be typeset in a small size, we can say, e.g.,

```
\def\footnote#1#2{#1\insert\footins
  \noindent#1\sevenrm#2}}
```

which typesets the footnote text in seven-point roman. The footnote symbol will be set in the current font (the font that is current at the time of insertion).

Readers experimenting with these macros will notice that the two examples of `\footnote` above result in bad vertical spacing, both inside and between the footnotes. The reasons are (1) selecting a font does not automatically change the interline spacing. The value of `\baselineskip` in `\box\footins` is still `12pt`, appropriate for cmr10, but not for cmr7; (2) there is no separation, in `\box\footins`, between the individual footnotes.

To correct the spacing, (1) the interline glue (`\baselineskip`) should be set, in `\box\footins`, to a value appropriate for a seven-point font; (2) the individual footnotes should be separated by placing a strut with the desired height and depth at the beginning and end of each of them (see also [Ex. 21.3]). Much better footnote spacing is obtained by:

```
\def\footnote#1#2{#1\insert\footins{
 \baselineskip=8pt\noindent\sevenrm
 \strut#1#2 \strut}}
```

Further improvement is obtained when TeX is discouraged from splitting a footnote between pages, whenever possible. This is done by (1) placing a penalty between the lines of each footnote; (2) placing a negative penalty between footnotes in `\box\footins`; (3) adding flexibility to the `4pt` separating the footnotes. Some flexibility may also be added to the interline glue, but this results in nonuniform appearance of the pages.

```
\def\footnote#1#2{#1\insert\footins{
 \interlinepenalty=1000
 \baselineskip=8pt plus1pt\noindent
 \sevenrm#1#2\endgraf \penalty-1000
 \vskip4pt plus2pt minus2pt}}
```

The last point to consider is the two parameters `\leftskip`, `\rightskip`. They are inserted on the left and right of every line of text [100]. Normally they are zero, but the user may set them to any value at any time. If we don't want them to affect the horizontal size of our footnotes, they should be set to zero locally, when the footnote text is inserted into `\footins`. This is done by:

```
\def\footnote#1#2{#1\insert\footins{
 \leftskip=0pt\rightskip=0pt
 \interlinepenalty=1000
 \baselineskip=8pt plus1pt\noindent
 \sevenrm#1#2 \penalty-1000
 \vskip4pt plus2pt minus2pt}}
```

3. The OTR should ship out a page consisting of (1) the body of the text, in `\box255`; (2) a `\vskip\skip\footins`, with a rule *inside it*; (3) the footnotes for the page, in `\unvbox\footins`. Here is how it's done:

```
\output={\shipout\vbox to \vsize{\unvbox255
  \ifvoid\footins\else
    \vskip\skip\footins
    \kern-3pt\hrule width2in\kern2.6pt
    \unvbox\footins
  \fi}
  }
```

In practice, the OTR should do other things, such as typesetting and incrementing the page number, but those are ignored here. The reason for unboxing `\box255` is so that its flexible glues could blend with the ones in the insertion box (see the last six lines on [125] for a similar comment).

It should be mentioned here that these footnotes may appear on a page different from the one on which they are referenced (see [Ex. 15.13] for other cases where this may happen). This happens when there are many footnotes but we limit the amount of space on the page where footnotes can be typeset by assigning a small value to `\dimen\footins`. A value such as `0.4in` is generally enough for 4 footnote lines and, if there is more footnote text for the page, it would be typeset on the following page. This sometimes requires splitting a footnote into two parts, which is why footnotes should be inserted into `\footins` as individual lines, not as a `\vbox` (which is indivisible). Thus we should avoid something like:

```
\def\footnote#1#2{#1\insert\footins{
  \vbox{#1#2 \vskip4pt}}}
```

Example: Right margin insertions

Another useful example of insertions has to do with index items. Preparing an index for a textbook can be no small task, and TeX can help a lot in this (Note 12). Typically, macros should be defined to identify parts of the text as index items, and write them on a file for future sorting and processing. However, it is very useful, while writing and modifying the document, to typeset all the index items of a page on the right margin of the page. When the document is ready, the final run omits the notes on the margin. Such an example is shown on [415, 423–424] and is described here in a simplified form.

The main steps are:

1. A boolean variable `\proofmode` is declared and set to `true`. A new class of insertions, called `\margin`, is declared and initialized.

```
1. \newif\ifproofmode
2. \proofmodetrue
3. \newinsert\margin
4. \dimen\margin=\maxdimen
5. \count\margin=0
6. \skip\margin=0pt
```

Line 4 allows any amount of marginal notes per page (Note 13). Line 5 guarantees that no space will be reserved on the page for the notes, and line 6 says not to skip vertically before the notes are typeset.

2. The index macro is defined. It has *one* parameter, the index item. The macro writes it on a file, with the page number, and inserts it in `\insert\margin`. The latter part is done by:

```
\ifproofmode\insert\margin{
 \hbox{\sevenrm #1}}\fi
```

Each index item is placed in an `\hbox`, and so becomes one line. If it is too long to fit on the margin, part of it will fall off the page. If it is important to see the entire text of the note, it can be placed in a narrow `\vbox`, where it will be broken into lines. Assuming a 1 inch wide margin, we can write:

```
\ifproofmode
 \insert\margin{\vbox{\hsize=1in
   \baselineskip=8pt\tolerance=2000
   \sevenrm\noindent#1}\smallskip}
\fi
```

Note the vertical spacing of the notes, which is similar to the case of footnotes. (Note 14)

3. The OTR should typeset `\box\margin` on the right margin of the page during `\shipout`. Here are the basic steps:

```
\output={\shipout\vbox to \vsize{
  \ifvoid\margin \else
    \rlap{\kern\hsize\kern4pt
      \vbox to0pt{\box\margin\vss}}
  \fi
  \unvbox255}}
```

The `\rlap` leaps over to the right margin with the `\kern\hsize`, then moves another `4pt` to the right, to separate the marginal notes from the body of the text. The OTR should, of course, do other things, such as advancing the page number, and appending a header, a footer, and footnotes.

The main differences between the marginal notes and the footnotes discussed earlier are (1) no marginal notes should be held over to the next page (even if they don't all fit on the current page); (2) no room should be reserved on the page for the marginal notes; (3) overfull boxes are okay since the marginal notes will be omitted anyway on the final run.

Example: Floating insertions

We describe a mechanism for floating insertions, similar to the `\midinsert` of the `plain` format. `\midinsert` is explained on [116] and its definition shown on [363]. Our example is simpler and does not do as much as `\midinsert`, but it works, and it serves to illustrate the principles involved.

An insertion class `\midins` is declared, and a macro pair `\midinsert`, `\endinsert` is defined and used to delimit the material to be inserted. It is used as follows:

```
\midinsert
⟨material to be inserted⟩
\endinsert
```

The material to be inserted may contain commands and specifications that should be kept local to the insertion (Note 15). This is achieved by the `\bgroup`, `\egroup` pair (see below), which acts as a quarantine. The main task of this pair, however, is to collect all the material appearing between `\midinsert` and `\endinsert`, and either typeset it, or place it in `\midins`. The `\begingroup`, `\endgroup` pair serves to localize the settings of `\box0` and `\dimen0` which the user never sees.

Most of the work is done by `\endinsert`. It closes the insertion material into `\box0`, and measures — with the help of our old friend, macro `\pagespace` — the amount of space left on the current page. If there is enough space, it typesets `\box0` immediately, otherwise, it inserts it in `\midins`.

```
\newinsert\midins
\dimen\midins=\vsize
\newdimen\spaceleft

\def\pagespace{...}

\def\midinsert{\par\begingroup
 \setbox0=\vbox\bgroup}
\def\endinsert{\egroup % finish the \vbox
 \pagespace
 \dimen0=\ht0 \advance\dimen0 by\dp0
 \ifdim\dimen0>\spaceleft
   \insert\midins{\unvbox0}
 \else
   \box0 \bigbreak
 \fi
\endgroup}
```

```
\output={\shipout\box255
 \ifvoid\midins\else\unvbox\midins\fi
 \advancepageno}
```

Note that \unvbox0, not \box0, gets inserted in \midins. This way the insertion material is the contents of \box0 and, if it is too large, TEX will be able to split it (unless it is itself a box). Also, the \unvbox\midins has the effect of placing the contents of \midins as a top insert at the head of an otherwise empty MVL.

The maximum size of inserted \midins material per page is the value of \dimen\midins which, in our case, is \vsize. This means that the entire page can be devoted to \midins insertions. However, if we set \dimen\midins=2in then each page will contain at most 2 inches worth of material from \box\midins. If \box\midins contains more than \dimen\midins of material, some of it will be held over to the next page (requiring, perhaps, splitting one block of insertion material).

If the contents of \box0 is another box, then it is indivisible, and TEX will not split it. In such a case, more than \dimen\midins worth of material may appear on a page. In fact, the resulting page may even be larger than \vsize, and no error message would be issued. Thus when using unsplittable insertions, the user should make sure that they are not too big. A detailed discussion of insertion splitting appears later.

The appearance of the text can be improved if we automatically add some glue, such as a \bigskip, after each insertion. If a page is broken between the insertion and the glue, the glue will, as usual, be discarded at the top of the new page. Also, the natural size of the \bigskip, 12pt, should be included in the test for space left. Only \endinsert needs to be modified.

```
\def\endinsert{\egroup % finish the \vbox
 \pagespace
 \dimen0=\ht0 \advance\dimen0 by\dp0
  \advance\dimen0 by \bigskipamount
 \ifdim\dimen0>\spaceleft
   \insert\midins{\unvbox0 \bigskip}
 \else
   \box0 \bigskip
 \fi
\endgroup}
```

Readers experimenting with these macros will discover very quickly that insertions are sometimes typeset in reverse order. This may occur when a large insertion appears close to the bottom of a page. Imagine a situation where 3 inches are left on the page and the user calls \midinsert to insert a 4-inch-tall figure. \endinsert will save the figure in \midins and it will eventually appear at the top of the next page. Imagine now that the user immediately calls \midinsert to insert another figure, only 2 inches tall. Since there is room on the current page for the second figure, it will be inserted in place, with the result that the two figures are now inserted in reverse order.

A simple (but, unfortunately, incomplete) solution is: A new boolean variable, \ifSaved, is declared. When an insertion is placed in the insertion box, \endinsert invokes the OTR temporarily, using a penalty of −10001. The OTR sets \ifSaved to true, and returns without shipping out anything. When the OTR is invoked normally, it sets \ifSaved to false. \ifSaved therefore indicates whether an insertion has been saved on the current page.

When \endinsert finds that there is room on the current page for the current insertion, it typesets it only if \ifSaved is false.

```
\newif\ifSaved \Savedfalse

\def\midinsert{\par\begingroup
 \setbox0=\vbox\bgroup}
\def\endinsert{\egroup % finish the \vbox
 \pagespace
 \dimen0=\ht0 \advance\dimen0 by\dp0
 \ifdim\dimen0>\spaceleft
   \insert\midins{\unvbox0}\penalty-10001
 \else
   \ifSaved
     \insert\midins{\unvbox0}
   \else
     \box0 \bigbreak
   \fi
 \fi
\endgroup}

\output={%
  \ifnum\outputpenalty=-10001
   \global\Savedtrue
   \unvbox255
  \else
   \global\Savedfalse
   \shipout\box255 \advancepageno
   \ifvoid\midins\else\unvbox\midins\fi
  \fi}
```

This is a good solution that works almost always. It may fail in some rare cases, however. The reason is that \penalty-10001 does not invoke the OTR immediately. The penalty is stored in the MVL, and is only noticed by TEX when it starts looking for a good point to break the page. This process is explained in detail in part II, but here is an example.

Imagine a case where there is an insertion, with `\penalty-10001`, on line 60, and page 7 should be broken around that line. When TeX invokes the page break algorithm, it notices the special penalty, breaks the page at that point, and invokes the OTR. The OTR also senses the special penalty and assumes that there is an insertion on page 7. The OTR then returns the material to the MVL, which causes TeX to immediately start looking for a page break. Since the special penalty is no longer there (Note 16), TeX may select a different breakpoint, such as line 59. Line 60 is now the first line of the next page, page 8, but the OTR has already assumed that there is an insertion on page 7.

`\topinsert` and `\pageinsert`. These macros are part of the `plain` format, in addition to `\midinsert` [115–116]. Material appearing between `\topinsert` and `\endinsert` is considered a *floating top insertion.* TeX will try to place it at the top of the current page but, if there is not enough room on the current page, the material will be placed at the top of the next one. Similarly, material appearing between `\pageinsert` and `\endinsert` is stretched to the size of a page, and becomes the next page.

Readers who have read the preceding text and examples are urged to look at [363] and try to understand the definitions of the three macros.

Example: Two insertion classes

It is possible, of course, to declare several insertion classes and limit the amount of insertions placed on a page from each class. Following are the outlines of a case where two insertion classes, `\midins` and `\footins` are declared and limited to `2.5in` and `1in` per page, respectively.

```
\newinsert\midins \newinsert\footins
\dimen\midins=2.5in
\skip\footins=12pt plus4pt minus4pt
\count\footins=1000
\dimen\footins=1in

\def\midinsert{...}
\def\endinsert{...}
\def\footnote#1#2{...}

\output={\shipout\vbox{\box255
  \ifvoid\footins\else
   \vskip\skip\footins
   \kern-3pt\hrule width2in\kern2.6pt
   \box\footins
  \fi}
  \ifvoid\midins\else\unvbox\midins\fi
  \advancepageno}
```

The OTR ships out `\box255` followed by the footnotes, and TeX's insertion mechanism guarantees that the total amount of footnotes will not exceed `1in` per page. Also, if there are `\midins` insertions, they will not exceed `2.5in` per page.

It is now clear why material is not inserted directly into the insertion box but is saved in a temporary buffer. This is how insertion material can be held over for the next page. Right before the OTR is invoked, the right amount of material is moved from the buffer and is placed in the insertion box.

The `plain` format OTR

Short and elegant, this OTR makes a good example, since it supports both footnotes and floating insertions. It is described on [255–256] and, therefore, only a few short remarks are necessary here. The first step is to define a macro `\plainoutput`

```
\def\plainoutput{\shipout\vbox
    {\makeheadline\pagebody\makefootline}
  \advancepageno
  \ifnum\outputpenalty>-20000
    \else\dosupereject\fi}
```

following which, the OTR is defined by

```
\output={\plainoutput}
```

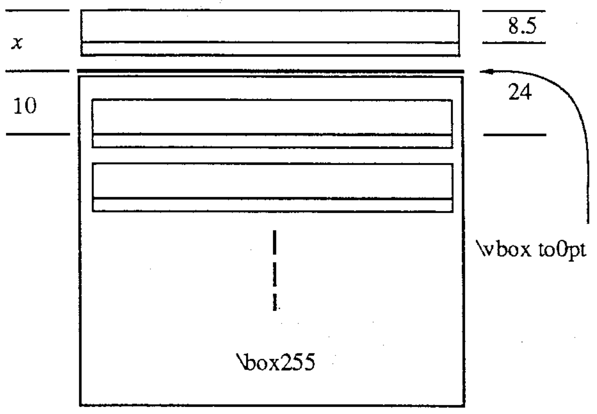
This way, the OTR can be redefined and then reset back to its original definition.

```
\def\makeheadline{\vbox to0pt{\vskip-22.5pt
  \line{\vbox to8.5pt{}\the\headline}\vss}
  \nointerlineskip}
```

Macro `\makeheadline` is the first item shipped. It suppresses the normal interline glue, so it is placed right on top of the second item (which is supplied by `\pagecontents`, see below). To achieve a uniform appearance of the document, the headline should have the same position, relative to the main body of the text, on all the pages. Its baseline is positioned, by `\makeheadline`, exactly 24pt above the baseline of the top line of `\box255`. This is achieved by placing the headline in a `\vbox to0pt`, moving up 22.5pt in the box, and typesetting the headline. The quantity 22.5pt (see diagram on following page) is the value that x should have in order that $x + 10$ should be equal to $24 + 8.5$.

The quantity `\headline` is declared as a `\toks` variable by `\newtoks\headline` and is set to an empty line `\headline={\hfil}`. It can be reset by the user to any token string.

Macro `\pagebody` limits the depth of the page to the value of parameter `\maxdepth`, whose `plain` format value is 4pt [348]. (See discussion of `\boxmaxdepth` in part I. See also the section, later in this part, on the depth of the current page.)



Position of Headline

```
\def\pagebody{\vbox to\vsize
 {\boxmaxdepth=\maxdepth \pagecontents}}
```

The `\pagecontents` macro starts by preparing the floating insertions, if any. It then opens `\box255` and, finally, prepares the footnotes, if any.

```
\def\pagecontents
 {\ifvoid\topins\else\unvbox\topins\fi
 \dimen0=\dp255 \unvbox255
 \ifvoid\footins\else
   \vskip\skip\footins
   \footnoterule
   \unvbox\footins
 \fi
 \ifraggedbottom \kern-\dimen0 \vfil \fi}
```

The two insertion boxes and `\box255` are opened, exposing their glues. The glues are now flexed to help `\pagebody` prepare a `\vbox to\vsize`. If the user wishes a ragged bottom, a `\vfil` glue is placed at the bottom of the page. This glue is flexed together with the other flexible glues on the page, leaving a glob of glue of non-zero size at the bottom of the page. The result is pages in which the bottom lines are not all at the bottom of the page. It should be noted that the definition of `\raggedbottom` (on [363]) also makes the `\topskip` glue stretchable, and that there is a `\normalbottom` macro (defined on the same page) that cancels the ragged bottom effect.

Macro `\footnoterule` creates the rule separating the footnotes from the main body of the text. The rule is placed 3pt above the top footnote.

```
\def\footnoterule{\kern-3pt
  \hrule width 2truein \kern 2.6pt}
% the \hrule is .4pt high
```

Finally, macro `\makefootline` places the footline 24pt below the main body of the page.

```
\def\makefootline{\baselineskip=24pt
 \line{\the\footline}}
```

The footline itself is a `\toks` variable declared by `\newtoks\footline`, and is set to

```
\footline={\hss\tenrm\folio\hss}
```

The page number. Some of the information in this section has already appeared in part I, and is repeated here for the sake of completeness.

In book publishing, both roman and arabic numerals are used for page numbers. Variable `\count0` is reserved by the `plain` format for the page number (`\countdef\pageno=0`) and, consequently, should not be used for anything else. It is initialized to one (`\pageno=1`), and is handled by several useful macros:

```
\def\folio{\ifnum\pageno<0
 \romannumeral-\pageno
 \else\number\pageno \fi}
\def\nopagenumbers{\footline{\hfil}}
\def\advancepageno{\ifnum\pageno<0
 \global\advance\pageno by -1
 \else\global\advance\pageno by 1 \fi}
```

Macro `\folio` typesets the page number either in arabic numerals or, if it is negative, in roman numerals.

The `\nopagenumbers` macro suppresses page numbers by eliminating them from the `\footline`.

Macro `\advancepageno` increments the page number by either 1 or −1, depending on its sign.

In certain documents, composite page numbers are used, which consist of more than one number. A page number such as 12–52 is common and usually refers to page 52 of chapter 12. The best way to implement such numbers in TEX is to use some of the ten counters `\count0` through `\count9` [119, 254]. They should be declared, initialized, incremented and typeset by the user. TEX, however, helps in two ways:

• It writes the values of the ten counters on the dvi file with each page. This helps the preview program and the printer driver identify the pages previewed or printed. In fact, those programs do not know what page number actually appears on the page, and they consider the ten values on the dvi file as *the* page number. The user should thus refer to those ten numbers when communicating with any program that handles the dvi file.

• TEX also displays the ten counters on the user's terminal, with trailing zeros omitted, when a page is shipped out. This is how things such as `[1]`, `[12.0.52]` are displayed at typeset time.

`\supereject`

The `\bye` control sequence, which is the recommended way to stop, is a macro defined by

`\par\vfill\supereject\end`. Why `\supereject` and not just `\eject`? And what is `\supereject`?

If many insertions are used throughout a document, there is a good chance that, after the last page is shipped out, some insertions will be left in their buffers, waiting to be typeset. This should be done as part of the 'end game' of TEX, which is initiated by the `\supereject` macro [116].

It is defined on [353] as `\par\penalty-20000`. The `plain` format output routine tests (on [255]) for this value and, if `\outputpenalty=-20000`, expands macro `\dosupereject`. This macro, defined on [256], tests the parameter `\insertpenalties` (see below) to see if any insertions remain heldover in their buffers. If there are any, `\dosupereject` makes sure that the output routine will be invoked again, giving it a chance to shipout those insertions. To make sure that the OTR is invoked again, `\dosupereject` prepares a blank page in the MVL by executing `\line{}\vfill\supereject`. This generates vertical material with a blank line at the top and a penalty of −20000 at the bottom. The material is simply left in the OTR(more precisely, put on the vertical list constructed by the OTR), which means it will be returned to the MVL, causing TEX to invoke the OTR again.

When the OTR is invoked again, it will output another page and, as usual, place `\topskip` worth of glue on top of it. To cancel that glue, `\dosupereject` really generates:

```
line{}\kern-\topskip\nobreak
\vfill\supereject
```

[256], but this is a minor point.

If there are any insertions left, they will be placed in their boxes each time the OTR is invoked for an empty page. The amount of inserted material per page is controlled, as usual, by the `\dimen` variable associated with the insertion.

A simple example is the endnotes described earlier. In that example, notes are accumulated in a temporary buffer, and should be typeset at the end of the document. This has to be done from the OTR, and the best way to do it is to use the special penalty generated by the `\bye`.

`\insertpenalties`

This is one of many *internal quantities* that TEX uses (see the complete list on [271]). During an OTR, it is equal to the total number of heldover insertions [254] (Note 17). A heldover insertion is an insertion (a parameter of an `\insert` command) that should have been typeset on the current page but, because of lack of space on that page did not make it, and will be made available to the OTR in the next page. Such a heldover insertion is sometimes split and only part of it appears on the current page.

Insertions (advanced)

This is advanced material, potentially useful to users who are heavily involved with OTRs and insertions, or to people who want a deeper understanding of TEX. For most users, however, the following quote (from [123]) may apply: "*On the other hand, maybe you don't really want to read the rest of this chapter at all, ever.*"

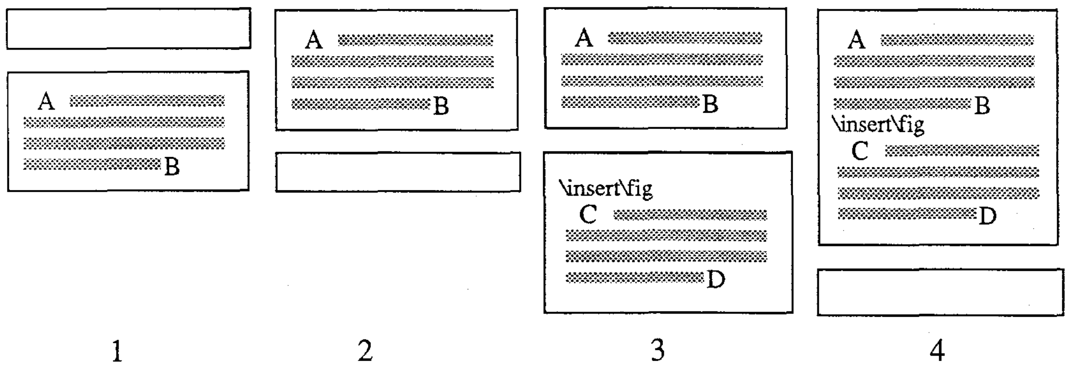
The current page and the list of recent contributions. As mentioned in part II, the MVL consists of two parts, the *current page* and, below it, the *list of recent contributions.* The current page holds the material that will become `\box255`. The recent contributions temporarily hold recently read material. After an entire paragraph has been read, it is typeset, and the lines of text appended to the recent contributions. At that point, the *page builder* is invoked (exercised). Its job is to move lines, one by one, from the recent contributions to the current page. For each line, the page builder calculates the cost of breaking the page after that line. For the first couple of lines the cost is very high because breaking there would result in a stretched page. Thus, for those lines, the badness b becomes 10000 and the cost c, 100000 (see formula on [111]).

At a certain point — when there are enough lines in the current page, for a normal page — b (and, as a result, c) starts getting smaller. A while later, there may be too many lines of text on the current page, and it has to be shrunk, increasing b and c again. The entire process can be seen, in real time, by setting `\tracingpages=1` [112]. If the page has to be shrunk more than its maximum shrinkability, both b and c become infinite. When c becomes infinite (or when a penalty ≤ -10000 is found, see below) the page builder goes back to the line of text where the cost was lowest, breaks the top of the current page and places it in `\box255` [§1017]. The bottom part of the current page is then returned to the recent contributions, and the page builder invokes the OTR.

The page builder is exercised at the end of a paragraph, at the end of a display equation within a paragraph, at the end of an `\halign`, and in a few other cases (see [122, 286]). The OTR can only be invoked by the page builder [§1025], which is why it is never invoked in the middle of a paragraph (unless the paragraph contains display math material).

The advanced reader might want to glance at [§980–1028] for the actual code of the page builder.

Since the page builder is exercised quite often, the list of recent contributions is usually small or



Figures. 1–4.

empty, and the current page gets larger and larger. When the OTR is invoked, the current page is empty. The `\showlists` command can always be used to display the two parts of the MVL in the log file.

The quantity t (`\pagetotal`) mentioned before as the height of the MVL is, actually, the height of the current page. It is updated by the page builder each time a line (or glue) is added to the current page.

A better understanding of this process must include glue and penalties. They are appended to the recent contributions, with the lines of text, when a paragraph is typeset, and are eventually moved to the current page. If the current page is empty, all glues, kerns and penalties moved to it are discarded. When the first box is moved to the current page, glue is added above it to keep its baseline `\topskip` below the top of the page. Following that, all glue, kern, and penalties are moved, with the text, from the recent contributions to the current page.

When a penalty ≤ -10000 is encountered, TeX breaks a page. The resulting page may be underfull. Such penalty values can be used to eject a page (by `\vfill\penalty-10000`), or to communicate with the OTR.

It should be stressed again, however, that `\penalty-10000` does not invoke the OTR *immediately*. If such a penalty is created inside a paragraph, between lines of text, it is saved in the recent contributions with the lines, and is only recognized as special when it is moved, by the page builder, to the current page. As a result, if a paragraph contains:

```
...\dimen0=2pt...\vadjust{\penalty-10000}
 ...\dimen0=1pt...\par
```

the OTR will be invoked after the entire paragraph has been read and broken into lines, and will find `\dimen0` to be `1pt`.

A page can be broken only at a glue, kern or penalty. If a page is broken at a glue or kern, the glue stays in the recent contributions (to be discarded when moved to the top of the next page). If the page is broken at a penalty, the penalty is saved in variable `\outputpenalty` and removed from the vertical list. This variable can be used to communicate with the OTR. Also, if the user wants to return some material from `\box255` to the current page, he may want to reinsert the penalty, by saying `\penalty\outputpenalty`.

Insertions and the page builder. We are now familiar with how the MVL is maintained in cases that don't involve insertions. In this section we see how insertions are handled in the MVL by the line break algorithm and the page builder. Let's assume that an insertion class n has been defined. When an `\insert`n is read from the source file, both the command and its insertion material are placed in the recent contributions. The next time the page builder is exercised, it finds the command, followed by the insertion material. The material should not be moved to the current page, since it is an insertion (review the definition of insertions). Instead, it should be moved to `\box`n, so the OTR should be able to typeset it anywhere on the page. However, material is only moved to `\box`n just before the OTR is invoked (see below). Therefore, when the page builder discovers the command, it (1) moves the command (and the insertion material), to the current page, but as a special item, not as a regular part of the current page (the material will later be moved to `\box`n from the current page); (2) decrements g by the size (height plus depth) of the insertion material.

Figures 1–2 show a paragraph (A–B) read into the recent contributions and moved to the current page. Figures 3–4 show how an `\insert\fig` command, followed by insertion material (C–D), is also read into the recent contributions and moved, as a special item, to the current page.

Splitting insertions. Before the page builder decrements g, it executes the rules on [123–124] to determine how much of the insertion material can appear on the page. If there is no room for the entire insertion — either because it is large, or because `\dimen` n has been assigned a small value — the rules tell how to determine a good point to split the insertion material so the remainder can be *held over* for the next page. The result obtained by the rules is used to decrement g, to reserve room on the page for the insertion.

Again, it should be emphasized that the split itself does not occur at this point. It takes place just before the OTR is invoked (see below). At that time, the top part of the split insertion is placed in `\box` n, and the bottom part is saved as a heldover insertion.

The rules for splitting insertions, in simplified form, are:

1. The first `\insert` n for the page decrements g by (the natural size of) `\skip` n, and again by the height plus depth of `\box` n. Note that g is not decremented by the size of the present insertion (this is done in rule 3.)

What can `\box` n contain at this point?

1a. It may be empty.

1b. It may contain material from the previous page. Typically, such material should have been typeset, by the OTR, on the previous page, and `\box` n emptied. However, if the OTR did not empty the box, room is now reserved for its contents on the present page.

1c. It may contain material placed there by the user explicitly, not through the `\insert` n command. In such a case, room is now reserved on the page for this material. If anything is placed explicitly in the box after this point, no room will be reserved for it on the page [§1009].

2. If a previous `\insert` n on the current page has been split (because it didn't fit on the page), the present insertion will certainly not fit on the page, and has to be held over. The only thing done at this point is to increment `\insertpenalties` by the parameter `\floatingpenalty`. This increases the cost of breaking the page at this point. See [124–125] for examples of values of `\floatingpenalty`.

3. Determine if the insertion will fit on the page without being split. If it will, decrement g by the size x (height plus depth) of the insertion material. Otherwise go to step 4 to calculate the split size.

We denote the quantity 0.001`\count` n by f. The value of g should be decremented by the scaled size xf of the insertion material.

An insertion will fit on the page if its scaled size xf is zero (or negative), or if

$$xf \leq g - t \qquad (1)$$

or if `\count` $n = 0$. The actual test also includes the `\pagedepth`, `\pageshrink` parameters, which are ignored here for simplicity. They are introduced in a later section.

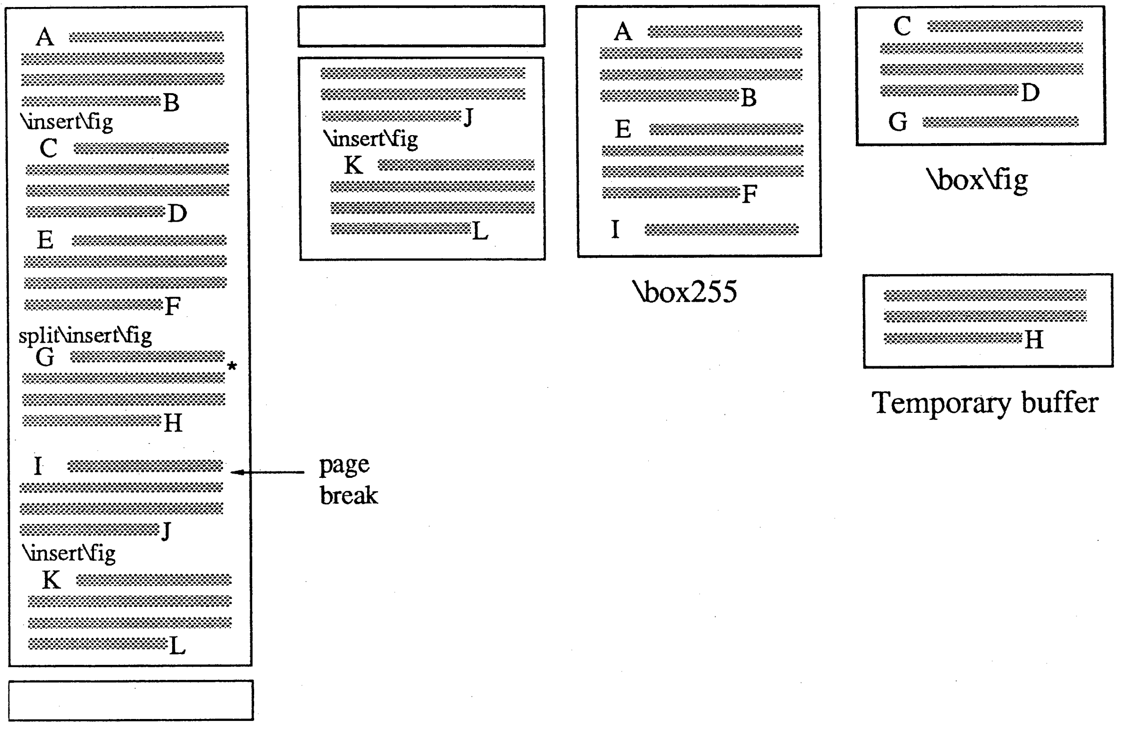
4. Determine where to split the insertion. Let's assume that we end up splitting `\insert` n at a distance v from its top. What determines v? After the material is split and is placed in `\box` n, the box's vertical size increases to $x + v$. The value of v should, therefore, be the largest number that satisfies (a) the new size, $x + v$, of `\box` n should be $\leq$ `\dimen` n; (b) v should also be $\leq g - t$ (the available space on the page). Relation (b) will also be modified later.

Since a split must occur between lines of text, it may be impossible to split `\insert` n to v. TeX therefore uses an algorithm, similar to the page builder but without insertions, to determine a value u close to v.

g is now decremented by u and the parameter `\insertpenalties` is incremented by the penalty value (if any) found at the split point. The page builder marks this insertion, in the current page, as a split insertion. Note that the split itself does not take place at this point. It is done after the page breakpoint is determined, and before the OTR is invoked.

All this happens when an `\insert` command is discovered by the page builder on the recent contributions, and is moved to the current page [§1000, §1008]. The page builder continues its operations and, finally, decides on a good breakpoint for the page. (Note: The value of `\insertpenalties` is used to help make the decision and, once it is made, `\insertpenalties` is free to be used for something else.) Fig. 5 shows an example of a current page with 3 paragraphs (A–B, E–F, and I–J) and 3 insertions (C–D, G–H and K–L) the second of which is stored in the current page as a split insertion (the '*' marks the split point.) The recent contributions list is empty.

The page builder then (see [125]) removes the bottom of the current page (everything below the breakpoint) and returns it to the recent contributions (Fig. 6). The next step is to place all the insertion material of class n in `\box` n. The page builder scans the current page and, for each `\insert` n found, appends the insertion material to `\box` n. When it finds a split insertion, it performs the actual split, appends the top part of the split material to `\box` n, and saves the bottom, as an independent insertion, in a separate place. All class n insertions found on the current page following this point, are saved in the same way, to be held over (Fig. 8).



5 6 7 8

Figures. 5–8.

For each of the heldover insertions saved, `\insertpenalties` is incremented by 1. This is why, in the OTR, this variable holds the number of heldover insertions.

The current page (without the insertion items) is now moved [§1017] into `\box255` (Fig. 7), which is set to height g, and the OTR is invoked [§1025]. It may return material to the MVL (to the recent contributions). When the OTR is finished, all the heldover insertions are moved from their saving place to the top of the recent contributions. At that point, if there is enough material in the recent contributions, TeX may exercise the page builder again.

If the OTR does not use the material in `\box` n, it stays in the box, and can be used in the following page. When the page builder builds the next page, g is decremented by the vertical size of `\box` n, just before the first block of insertion material is moved to the current page.

The d and z parameters in insertions. The two parameters `\pagedepth` and `\pageshrink` have already been mentioned in part I, in connection with the depth of boxes. They are denoted d and z, respectively, and are included in the tests for insertion split. The reader will recall Eq. 1: $xf \le g - t$, which means that an insertion of size x will fit on the page if its scaled size xf is less than or equal to the available room on the current page, $g - t$. Now, that we know about d and z, it is clear that we should include $t + d$, instead of just t, in this equation. The actual test performed by TeX [§1008] is

```
$\Delta = g - t - d + z$;
**if** $xf \le 0$ **or** $xf \le \Delta$
    **then** $g \leftarrow g - xf$
    **else** go to step 4.
```

Δ is the room left on the current page after it is shrunk as much as possible. f is the value of `\count` n (divided by 1000).

The last thing to be updated is rule 4*b* above. It states that a block of insertion material should be split at a distance v from its top, where v is the largest number that satisfies $v \le g - t$. The actual expression used [§1010] also takes d and f into account. It is $vf \le g - t - d$.

`\holdinginserts`. Sometimes the OTR is invoked temporarily, before the current page is completed, just so that it can examine the page so far and do something special. Such a special invocation is easy to do by saying `\penalty-10001`. The OTR can perform tests on `\box255` and return it to the MVL. If the OTR is invoked in such a way, it may be desirable to leave all insertion material in place and not put it in the insertion boxes. Starting with TeX version 3.0, this can be achieved by setting the new parameter `\holdinginserts` to a positive value.

This feature will be mentioned on [125] starting with the seventeenth printing of *The TEXbook.*

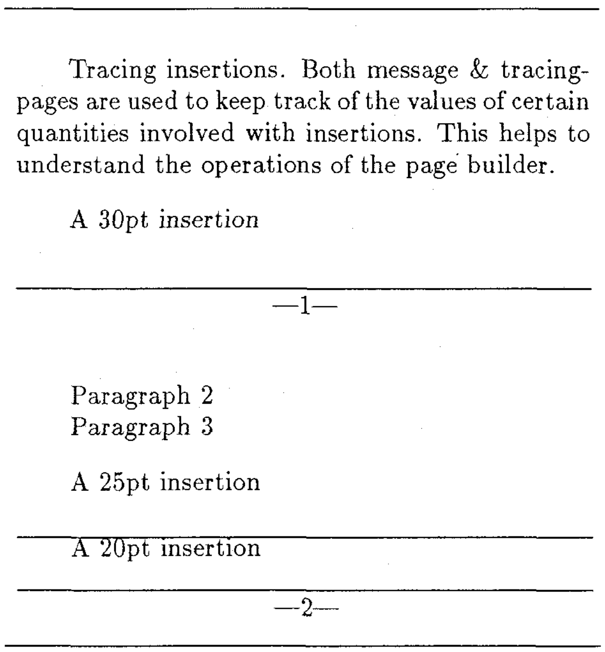
Tracing (in detail)

As mentioned before, a good way to learn about insertions is to trace the internal operations of TEX while it handles this 'sensitive' material. Fortunately, several tracing commands [303] are available, to bring out and print the values of many internal quantities. The most useful to us are `\message`, `\tracingpages` and `\showlists`. The following examples illustrate tracing, and should be studied, performed, and modified by the serious reader. This is an excellent way to understand the operations discussed in the previous section.

We start with a simple example involving 5 short paragraphs, and 4 unsplittable insertions.

```
\hsize=3in \vsize=100pt
\tracingpages=1
\showboxbreadth=1000 \showboxdepth=1

\newinsert\trace
\count\trace=1000
\skip\trace=12pt
\dimen\trace=100pt

\output={\message{%
   R: \the\ht255, \the\insertpenalties;}
  \shipout\vbox{\unvbox255
   \vskip\skip\trace \unvbox\trace
   \smallskip
   \centerline{\tenrm---\folio---}}
  \advancepageno}

\def\mes#1{\message{#1: \the\pagetotal,
 \the\pagegoal, \the\insertpenalties;}}

\mes1
Tracing insertions. Both message \&
tracingpages are used to keep track
of the values of certain quantities
involved with insertions. This helps
to understand the operations of the
page builder. \par\mes2

\insert\trace{\vbox to30pt{%
 A 30pt insertion\vfil\hrule}}
Paragraph 2 \par\mes3

\insert\trace{\vbox to25pt{%
 A 25pt insertion\vfil\hrule}}
Paragraph 3 \par\mes4

\insert\trace{\vbox to20pt{%
 A 20pt insertion\vfil\hrule}}
Paragraph 4 \par\mes5

\insert\trace{\vbox to15pt{%
 A 15pt insertion\vfil\hrule}}
Paragraph 5 \par\mes6

\bye
```

Typesetting the material above creates three small typeset pages (only the first two of which are shown here.)



Tracing insertions. Both message & tracingpages are used to keep track of the values of certain quantities involved with insertions. This helps to understand the operations of the page builder.

A 30pt insertion

—1—

Paragraph 2
Paragraph 3

A 25pt insertion

A 20pt insertion

—2—

It also generates the following log file.

```
1. \trace=\insert252
2. 1: 0.0pt, 16383.99998pt, 0;
3. %% goal height=100.0, max depth=4.0
4. % t=10.0 g=100.0 b=10000 p=250 c=100000#
5. % t=22.0 g=100.0 b=10000 p=0 c=100000#
6. % t=34.0 g=100.0 b=10000 p=150 c=100000#
7. 2: 46.0pt, 100.0pt, 0;
8. % t=46.0 g=58.0 b=10000 p=0 c=100000#
9. 3: 58.0pt, 58.0pt, 0;
10. % split252 to -1.94444,25.0 p=-10000
11. % t=58.0 plus 1.0 g=33.0 b=* p=0 c=*
12. R: 58.0pt, 0; [1]
13. %% goal height=100.0, max depth=4.0
14. % t=10.0 g=63.0 b=10000 p=0 c=100000#
15. 4: 22.0pt, 63.0pt, 0;
16. % t=22.0 plus 1.0 g=43.0 b=10000 p=0
17.   c=100000#
18. 5: 34.0pt, 43.0pt, 0;
19. % split252 to 7.05556,15.0 p=-10000
```

```
20. % t=34.0 plus 2.0 g=28.0 b=* p=0 c=*
21. R: 43.0pt, 0; [2]
22. %% goal height=100.0, max depth=4.0
23. % t=10.0 g=73.0 b=10000 p=0 c=100000#
24. 6: 22.0pt, 73.0pt, 0;
25. % t=22.0 plus 1.0 g=73.0 b=10000 p=0
26.   c=100000#
27. % t=23.94444 plus 1.0 plus 1.0fill g=73.0
28.   b=0 p=-20000 c=-20000#
29. R: 73.0pt, 0; [3]
```

Message 1 (line 2) shows the values of t and g before TEX encounters any text. Line 3 (with %%) shows the goal height, which is still `\vsize`. Line 4 is generated when the first text line is moved to the current page. It shows $t = 10$ pt, the height of the first line of text (plus the `\topskip` glue above it). Line 5 shows $t = 22$ pt, which is the height of the first text line, plus the `\baselineskip` following it, plus the height of the second line of text (the depth of the last line is the depth of the page, and is therefore not included in t). Lines 6–8 show t growing in steps of 12 pt until it reaches 46 pt, the total height of the 4 lines of the first paragraph. Message 2 (line 7) shows $t = 46$ pt and $g = 100$ pt, still equal to `\vsize`. However, line 8 shows that g was decremented, as a result of the first `\insert`, from 100 to 58, a difference of 42 pt. This equals the size (30 pt) of the material inserted, plus the natural size (12 pt) of `\skip\trace`.

Message 3 (line 9) shows $t = 58$ pt, because the second paragraph (a single line) was read, typeset, and moved to the current page. At this point both t and g equal 58 pt (but for different reasons!). It would seem like an ideal point to break the page, but the page builder starts looking for a page break only when $c = \infty$ or when the current penalty ≤ -10000 [§1005]. So it reads the next item from the source file, which happens to be the next insertion (25 pt). The page builder tries to move it to the current page, and it executes the 4 steps on [123–124]. Steps 1, 2, don't apply. The test in step 3 is not passed, so the page builder goes to step 4 and calculates a good splitting point for the insertion. The test on the second line of [124] results in $v = -d$ (since $t = g$ and $f = 1$). This means that the ideal split is at a point 1.9444 pt *above* its top. This is why line 10 shows that the page builder has tried to `split252 to -1.94444`. This is a strange split but, in any case, it cannot be done since the insertion is a box. The page builder thus moves the entire insertion to the current page, and decrements g to 33 pt.

However, the 58 pt of material cannot be shrunk to 33 pt, resulting in line 11 with `b=* p=0 c=*`, infinite badness and cost. This is the time to start looking for a page break, so the page builder goes back to the point, in the current page, with the least cost, and breaks the page there. What is that point? The current page contains 5 lines. Each of the first 4 lines is associated with a cost of 100000, and the last line has infinite cost. The most logical point for a page break is, therefore, following the fourth line.

The part of the current page below the breakpoint (consisting of the line "Paragraph 2" and the 25 pt insertion) is returned to the list of recent contributions. The insertion material from the current page is moved to `\box\trace`, the rest of the current page is moved to `\box255` (actually, the rest of the current page *becomes* `\box255`), and the page builder invokes the OTR.

A `\showlists` command placed in the OTR would show no current page, and recent contributions consisting of the line "Paragraph 2" and the 25 pt insertion.

The R message (line 12) shows `\ht255` = 58 pt, so the total height of the page shipped out is $58+12+30 = 100$ pt. This is a successful case since, with many unsplittable insertions, some pages must be stretched a lot.

The next page starts with (line 14) $t = 10$ pt (one line of text, "Paragraph 2"), and $g = 63$ pt ($= 100-12-25$). On lines 16–18 t is incremented to 34 pt, which means that 3 lines of text (paragraphs 2, 3 and 4) are tentatively considered). Message 5 (line 18) shows $g = 43$ pt which means that the 20 pt insertion has been read. It also shows $t = 34$ pt which means that there is still room on the page for 9 pt worth of material (typically 7 pt high and 2 pt deep).

The next item is read from the source file. It is the 15 pt insertion. the page builder calculates (line 19) a split point (`split252 to 7.05556`) but, since it is an (indivisible) box, it cannot be split. It is moved to the current page, causing an infinite cost (line 20). A page break point is determined as before, and it is following the second line ("Paragraph 3"). Paragraph 4 and the 15 pt insertion are returned to the list of recent contributions, and the current page becomes `\box255`.

The R message (line 21) shows `\ht255` = 43 pt. The box contains just two lines of text (a height of 22 pt) and was stretched to 43 pt at the paragraph break.

The rest of the log file, pertaining to the third page, is easy to read and is left as an exercise.

Exercise: Add flexibility to `\skip\trace` (such as `12pt plus6pt minus4pt`) and typeset the example. Make sure that you see how the flexibility is reflected in the values for t.

Exercise: Change `\vsize` to 90 pt and repeat the experiment. The main changes should be in the splitting. The page builder will try to split the insertions at different points. Since the insertions are indivisible, they will not be split.

Exercise: Add `\showlists` commands after each `\message`, and in the OTR. You may have to fiddle with the values of `\showboxbreadth` and `\showboxdepth` in order to get the right amount of output.

The next experiment deals with splittable insertions. We modify the source file to:

```
\hsize=3in \vsize=100pt
\tracingpages=1
\showboxbreadth=1000 \showboxdepth=1

\newinsert\trace
\count\trace=1000
\skip\trace=12pt
\dimen\trace=100pt

\output={\message{R:
    \the\ht255, \the\insertpenalties;}
  \shipout\vbox{\unvbox255
    \vskip\skip\trace \unvbox\trace
    \smallskip
    \centerline{\tenrm---\folio---}}
  \advancepageno}

\def\mes#1{\message{#1:
 \the\pagetotal, \the\pagegoal,
 \the\insertpenalties;}}
\mes1

Tracing insertions. Both message \&
tracingpages are used to keep track
of the values of certain quantities
involved with insertions. This helps
to understand the operations of the
page builder. \par\mes2

\insert\trace{\noindent* This is the
first insertion, about four lines worth
of text. This would make it possible
for \TeX\ to split the insertion,
if necessary. Up until now our insertions
were unsplittable}
Paragraph 2 \par\mes3

\insert\trace{\noindent* This is the
second insertion, three lines worth
of text. This would make it possible
for \TeX\ to split the insertion, if
necessary.}
Paragraph 3 \par\mes4

\insert\trace{\noindent* The third
insertion, four lines worth of text,
to illustrate the insertion splitting
rules on [123]. Note how this is split,
and how the split part is typeset
following the text on this page.}
Paragraph 4 \par\mes5

\insert\trace{\noindent*
 Insertion 4, one line.}
Paragraph 5 \par\mes6
\bye
```

This produces 3 typeset pages, only the first 2 of which are shown here.

Tracing insertions. Both message & tracingpages are used to keep track of the values of certain quantities involved with insertions. This helps to understand the operations of the page builder.

* This is the first insertion, about four lines worth of text. This would make it possible for TeX to split the insertion, if necessary. Up until now our

—1—

Paragraph 2
Paragraph 3

insertions were unsplittable
* This is the second insertion, three lines worth of text. This would make it possible for TeX to split the insertion, if necessary.
* The third insertion, four lines worth of text, to il-

—2—

It also generates the following log file:

```
\trace=\insert252
1: 0.0pt, 16383.99998pt, 0;
%% goal height=100.0, max depth=4.0
% t=10.0 g=100.0 b=10000 p=250 c=100000#
% t=22.0 g=100.0 b=10000 p=0 c=100000#
% t=34.0 g=100.0 b=10000 p=150 c=100000#
2: 46.0pt, 100.0pt, 0;
% split252 to 40.05556,33.44444 p=150
% t=46.0 g=54.55556 b=10000 p=0 c=100000#
3: 58.0pt, 54.55556pt, 150;
% t=58.0 plus 1.0 g=54.55556 b=* p=0 c=*
R: 54.55556pt, 1; [1]
%% goal height=100.0, max depth=4.0
% t=0.0 g=76.05556 b=10000 p=0 c=100000#
% t=10.0 g=42.61111 b=10000 p=0 c=100000#
4: 22.0pt, 42.61111pt, 0;
```

```
% split252 to 18.66667,9.44444 p=250
% t=22.0 plus 1.0 g=33.16667 b=10000 p=0
  c=100000#
5: 34.0pt, 33.16667pt, 250;
% t=34.0 plus 2.0 g=33.16667 b=* p=0 c=*
R: 33.16667pt, 1; [2]
%% goal height=100.0, max depth=4.0
% t=0.0 g=52.05556 b=10000 p=0 c=100000#
% t=10.0 g=42.61111 b=10000 p=0 c=100000#
6: 22.0pt, 42.61111pt, 0;
% t=22.0 plus 1.0 g=42.61111 b=10000 p=0
  c=100000#
% t=23.94444 plus 1.0 plus 1.0fill
  g=42.61111 b=0 p=-20000 c=-20000#
R: 42.61111pt, 0; [3]
```

The main differences between this experiment and the previous one are:

1. Insertions can now be split. The message `split252 to 40.0555,33.4444 p=150` shows that the first insertion should, ideally, have been split at a distance of 40 pt from the top. Such a point, however, is between two lines of text, so the insertion ended up being split at 33.4 pt, after the third line of text. Note the widowpenalty of 150 found there.

The split operation is similar to a page break, a fact which shows us how to control insertion splitting. We can, e.g., place a penalty of −10000 in the first insertion.

```
\insert\trace{\noindent* This is the first
insertion, about four lines worth of text.
\vadjust{\penalty-10000} This would make
it possible ... were unsplittable}
```

This will force a split of the insertion after the second line. The log file will now contain the line:

```
split252 to 39.5,21.6527 p=-10000
```

showing that the split occurred 21.6 pt from the top (a height of two lines) because of the large negative penalty found.

2. The displayed values of `\insertpenalties` show the dual nature of this parameter. Several messages display the value 150 (= `\widowpenalty`). In the OTR, however, the value of `\insertpenalties` is not a penalty but the number of heldover insertions. When the first page is shipped out, the second insertion has already been read, and is being held over, together with the split part of the first insertion. As a result, the value of `\insertpenalties` in the OTR is 2.

Exercise: Place `\showlists` commands after each `\insert\trace{...}` and in the OTR. This will show how inserted material is stored in the recent contributions and in the current page.

Summary

This is a tutorial, not a cookbook. It does not contain any canned macros that can be directly copied and used. Instead, it tries to develop a better understanding of insertions, so that the reader will be able to implement insertions for specific applications.

All the material presented here (except, perhaps, some examples) can be found in *The TEXbook*, although in a somewhat cryptic language. The serious reader should, therefore, after reading this tutorial and doing the exercises, go back to the book to get a different perspective on the topics discussed here.

Endnotes

[1] This is an endnote. Look at the endnotes example to see how it works.

[2] The idea is that, when a textbook is written, items that should appear in the index of the book should be flagged by the author and written by TEX on a file, for the future preparation of an index. While the book is being written and proofread, it is also handy to have all the index items for a page printed on the right margin of that page. On the final printing of the page, those items are suppressed.

[3] Given two large figures that are textually related, they should be inserted into the document close to each other. If they don't both fit on one page, they should be inserted on facing pages, which means that the first figure should be inserted on the next even-numbered page, and the second figure, on the page following.

[4] All that the user has to do is save the figures in boxes and check, in the OTR, for the next even-numbered page.

[5] Answer: Because glue and kern are discardable items and disappear at a page break.

[6] Actually, in a temporary place.

[7] Actually, just before the OTR is invoked, the material is brought in from temporary storage and is appended to the box. Note that the allocated box may contain other material, placed there by the user not through the `\insert` command. Such material remains in the box and is eventually typeset on the page by the OTR. However, no room is reserved on the page for such material, and it may cause a page overflow.

[8] Actually, in a temporary buffer.

[9] We use t to denote `\pagetotal`, and g to denote `\pagegoal`.

[10] The temporary buffer is appended to it.

[11] The `\vsplit` command works by splitting a vbox at a permissible point. If the insertion material is made up of line boxes, it will be split *between lines*, not in the middle of a line. Penalties also control the split. Sometimes a box will be split at a point away from where we wish, because of a penalty that encouraged breaking the box at that point. However, the material split will be shrunk or stretched to bring it to the desired size.

[12] Although it cannot do the entire job.

[13] If the amount of marginal notes exceeds `\vsize`, some of it will be printed off the page, but will not be held over to the next page.

[14] Because of the narrow box width, there will be overfull boxes, but the thick vertical bars accompanying them can be eliminated by `\overfullrule=0pt`.

[15] Things like `\hsize=`*xxx*, `\raggedright`, and `\obeylines`.

[16] It is not returned to the MVL when the OTR says `\unvbox\midins`.

[17] However, outside the OTR it contains, not the number, but the sum of penalties, of all the heldover insertions [111].

◇ David Salomon
California State University, Northridge
Computer Science Department
Northridge, CA 91330
`dxs@mx.csun.edu`

Macros

A New Editor

Victor Eijkhout

Starting this issue, I've joined the editorial committee as associate editor for macro affairs (see the reverse of the title page for the other members).

The fact that incoming articles about TeXnical affairs will undergo my scrutiny does not mean that there is suddenly a large chance that submitted articles will be returned, rubber-stamped 'rejected'. My job will be to assist authors in creating articles that are of maximum value to the *TUGboat* readership. Often this means that my main concern is 'how well does this article explain whatever it is telling', rather than 'is this all completely original'. Remember that TeX is not something you read about, it is something you actually *do*. The subject matter of the article is therefore a secondary concern: *TUGboat* is read by beginners and grand masters alike, so articles need not be very high-brow. In fact, we need more articles that help the beginners take the first steps to grand masterhood.

Let these few lines with which I have introduced myself then also be an invitation to prospective authors: if you have done something new, or if you have something interesting to say about something old, write it down, and send it to *TUGboat*. Should you have trouble with the finishing touch, send in what you have and we will discuss it.

◇ Victor Eijkhout
Center for Supercomputing Research and Development
University of Illinois
305 Talbot Laboratory
104 South Wright Street
Urbana, Illinois 61801-2932, USA
`eijkhout@csrd.uiuc.edu`

Line Breaking in \unhboxed Text

Michael Downes

In the course of my work (macro writing and troubleshooting for TeX-based production at the American Mathematical Society) I recently had to investigate a line-breaking problem in the bibliography macros of the documentstyle `amsppt`, used with AMS-TeX. This is a report on the results of my investigations. Applications where this information might be useful include (1) implementation in TeX of SGML-style macros with omitted end tags as an option, and (2) using the width of a piece of text to choose between two formatting alternatives.

The amsppt bibliography macros

Although they're less sophisticated than BibTeX, the `amsppt` bibliography macros are simple to use and provide a certain degree of style independence (which makes the `.tex` file more portable). They are designed to allow the individual parts of a reference to be specified in any order, with punctuation between the parts and other formatting supplied automatically. In addition, an individual field within the reference does not have an ending delimiter: `\paper` (used for article titles) does not have a matching `\endpaper`, and so on. In SGML terminology, these would be called structures with 'omitted end tags'. A typical reference looks like this:

```
\ref\key C1
\by B. Coomes
\book Polynomial flows, symmetry
  groups, and conditions sufficient
  for injectivity of maps
\bookinfo Ph.D. thesis
\publ Univ. Nebraska--Lincoln
\yr 1988
\endref
```

In TeX, the combination of omitted end tags and randomly ordered elements (possibly with some elements absent) is not easy to provide. If it were required that all the tags had to be present, and in the right order, one way of obtaining at least the appearance of omitted end tags would be to define each beginning tag as a macro with an argument delimited by the next tag:

```
\def\key#1\by{⟨process the argument⟩\by}
\def\by#1\book{⟨process the argument⟩
  \book}
\def\book#1\bookinfo{⟨process the
  argument⟩\bookinfo}
```

and so on. Another approach would be to use `^^M` (carriage return) as the ending delimiter; however, this would require the user to have each element on a separate line, and to add percent signs at the end of each line but the last if an element were more than one line long—not too impractical perhaps in the context of a bibliography situation, but trouble-prone in general.

The straightforward approach of having `\key`, `\by`, etc., be macros with an argument enclosed in braces would work fine, but doesn't seem to be a case of true omitted end tags since the closing `}` is necessary. And it's a little more work for the user to type the braces.

The **amsppt** bibliography macros take a different approach, using `\hbox\bgroup ... \egroup`. The definition of `\by`, for example, ends with

```
\setbox\bybox=\hbox\bgroup
```

and the `\egroup` to end the box is supplied by `\book`, or whatever tag follows next. This stores the author name in the box `\bybox`. Each part of the reference is similarly stored away in a box instead of being put on the page immediately. The `\endref` macro then unboxes all the boxes, using `\unhbox`, and sets them in a paragraph in the proper order.

This method avoids reading the text as a macro argument, and makes omitted end tags possible, because during the `\setbox` operation TeX actually typesets the text, expanding macros along the way. This is essential if TeX is to find the `\egroup` to close each box.

The problem

Interestingly, it seems that the **amsppt** documentstyle was used for more than five years before the line-breaking problem, which had been present from the beginning, was identified (by Barbara Beeton's eagle eye). Most likely, the problem did manifest itself occasionally during that time but was dismissed without investigation because it could be resolved easily by adding a `\linebreak`. What Barbara noticed was that, in the example given above, the compound "Nebraska–Lincoln" wasn't breaking properly at the end of a line. The best line break was definitely after the en-dash, but instead "Lincoln" was hanging over the right margin. After she pointed the bad break out to me and we did some experiments, it became clear that although hyphenation between letters was working as normal, after explicit hyphens the possibility of a line break was disappearing.

Horizontal lists

If you have the complete Chapter 14 of *The TeXbook* stored in the "non-volatile memory" of your brain then you probably already know the cause of the phenomenon we were seeing. For those of you who don't, I'll review some terminology and ideas.

All characters typeset by TeX are put into what is called a "horizontal list". Characters and other elements of a math formula or subformula are first processed as a math list, but they end up being transformed into ordinary horizontal list material: characters, boxes, glue, penalties.

To be more specific, the components of a horizontal list are:

(1) characters;
(2) glue (usually interword spaces)
(3) kerns (usually adjustments between letters);
(4) discretionary breakpoints (usually discretionary hyphens);
(5) penalties (encouraging or discouraging line breaks);

(6) boxes (`\vbox`es or `\hbox`es containing subsidiary vertical or horizontal lists);

and (7) a few other miscellaneous kinds of things not important in the current discussion.

For line-breaking purposes TEX does not discriminate between single characters, boxes, or rules — each of these treated as a box with a particular width — except that automatic hyphenation occurs only between letters (more precisely, characters with `\lccode` $\neq 0$; nonletters normally have an `\lccode` of 0).

Horizontal lists are constructed either in "horizontal mode" or in "restricted horizontal mode". The former is the mode used in making ordinary paragraphs; the latter is the mode used inside an `\hbox`. Actually the material of a paragraph also ends up in `\hbox`es, because a finished paragraph is just a stack of `\hbox`es separated by `\baselineskip` glue; but the horizontal list for a paragraph is different in a few significant respects from a horizontal list constructed in restricted horizontal mode.

Recall that TEX optimizes line breaks over an entire paragraph; the horizontal list of a paragraph is not broken up into separate lines until the `\par` or other paragraph-ending command has been reached. At that point TEX goes through the entire horizontal list of the paragraph and chooses line breaks based on the current values of `\hsize`, `\parfillskip`, `\rightskip`, `\leftskip`, `\parshape`, `\tolerance`, `\hyphenpenalty`, and other parameters. During the initial construction of the horizontal list, TEX adds certain things to help in the line-breaking process.

These items added to a horizontal list by TEX are not explicitly present in the input file, but are inferred by TEX based on the context. Many of the seemingly magical effects of TEX are accomplished this way: paragraph indentation is obtained by inserting an `\hbox` of width `\parindent` in the horizontal list; and one step in the process of automatic hyphenation is the addition of `\discretionary{-}{}{}` in the horizontal list at all the hyphenation points determined by TEX's hyphenation patterns. These items aren't ephemeral, they're really there in the finished list, and can be seen using `\showlists`.

Because these items are really present in the finished list, they use up box memory (part of TEX's main memory). Therefore TEX avoids adding items unnecessarily. Primarily this means that in *restricted* horizontal mode — in the making of an `\hbox` — where line breaking is not a possibility, breakpoint items such as `\discretionary`s or `\penalty`s that would be added in *un*restricted horizontal mode are omitted.

Since each piece of an **amsppt** reference is typeset using `\hbox`, breakpoints are not added by TEX to the enclosed horizontal list. This is not a problem *unless* you `\unhbox` the box and reuse the contained horizontal list to make a paragraph. But that's exactly what the **amsppt** bibliography macros do.

Examples

Here are some examples of output from the `\showlists` command, to make it easier to picture the structure of horizontal lists.

Characters and glue. The word "in", along with surrounding word spaces:

```
\glue 3.33333 plus 1.66666 minus 1.11111
\tenrm i
\tenrm n
\glue 3.33333 plus 1.66666 minus 1.11111
```

Each line corresponds to one item in the horizontal list. In TEX's eyes `\tenrm` — the name of the current font — is not a separate piece of the horizontal list, but an attribute of the character "i" or "n". The font attribute is displayed with each character for informational purposes.

For the numbers given here the units are points; thus an interword space in this particular font has a natural width of about 3.33 pt, with stretchability of 1.67 pt and shrinkability of 1.11 pt. The em-width of the font is 10 points, so as you can see, the values correspond to $1/3$ em, $1/6$ em, and $1/9$ em.

Kerns and ligatures. The kerns added by TEX in a horizontal list are related to ligatures in that both of them are dependent on the current font:

> If two or more [ordinary characters] occur in succession, TEX processes them all as a unit, converting pairs of characters into ligatures and/or inserting kerns as directed by the font information. (*TEXbook*, p. 286)

Take the word "mode", for example: it has a kern added between the o and the d, in the font `\tenrm` (cmr10).

```
\tenrm m
\tenrm o
\kern0.27779
\tenrm d
\tenrm e
```

And as an example of a ligature, consider the "ff" ligature in the word "off":

```
\tenrm o
\tenrm ^^K (ligature ff)
```

The ligature character resides in font position 13, which is the ASCII location of control-K. When TeX reads two consecutive "f"s, it replaces them with a single control-K character in the horizontal list, following the instructions in the ligature table for this particular font. Similarly, the ligature character for an en-dash in the same font resides in the ASCII position 123, so in the output of a `\showlists` command it's represented by a left brace:

```
\tenrm { (ligature --)
```

Discretionaries. The discretionary items added by TeX in a horizontal list are of two kinds. A plain `\discretionary` is added after every hyphen character or ligature formed from hyphen characters. So a more complete picture of an en-dash is as shown here (using the text "1–9"):

```
\tenrm 1
\tenrm { (ligature --)
\discretionary
\tenrm 9
```

As already mentioned, to accomplish automatic hyphenation, a discretionary hyphen (equivalent to `\discretionary{-}{}{}`) is added at every hyphenation point within words, according to TeX's internalized hyphenation patterns. But this second kind of `\discretionary` is not added at the same time as the first kind. We'll see the significance of this shortly.

Penalties and glue. Penalties and glue added behind the scenes in a horizontal list are mainly added in math formulas. Internally, the automatic spacing in math formulas is done by adding `\glue` items in the horizontal list in the amount of `\thinmuskip`, `\medmuskip`, or `\thickmuskip`. Penalties in the amount of `\relpenalty` and `\binoppenalty` are added after binary relations and binary operators to allow line breaks. They serve essentially the same purpose as `\discretionarys`, but unlike `\discretionarys`, which neither encourage nor discourage a break, `\relpenalty` and `\binoppenalty` are usually set to some positive value that discourages line breaking. The plain TeX values are 500 and 700, respectively, so that line breaks after operators are discouraged slightly more than after relations. The horizontal list representation of the formula $a = b + c$ looks like this:

```
\mathon
\teni a
\glue(\thickmuskip) 2.77771 plus 2.77771
\tenrm =
\penalty 500
\glue(\thickmuskip) 2.77771 plus 2.77771
\teni b
\glue(\medmuskip) 2.22217 plus 1.11108
                          minus 2.22217
\tenrm +
\penalty 700
\glue(\medmuskip) 2.22217 plus 1.11108
                          minus 2.22217
\teni c
\mathoff
```

If `\mathsurround` were, say, 3pt instead of 0pt, the `\mathon` and `\mathoff` items would be followed by the note

```
(surrounded 3.0)
```

Mathsurround spacing behaves more or less like a kern of the given amount; if a line break occurs at the end of a math formula, the spacing is discarded to keep the line from ending short of the margin. If you look at the horizontal list using `\showlists`, you'll see that the `\mathoff` item remains, but the `(surrounded 3.0)` note disappears.

Automatic hyphenation

To summarize what's been covered so far: when TeX is working in restricted horizontal mode, it omits all the items that are needed only for line-breaking purposes — discretionaries after explicit hyphens, discretionary hyphens, and the penalties after math relations and binary operators. If the resulting horizontal list is then `\unhboxed` and used to make a paragraph, certain line breaks will simply be impossible because the breakpoints aren't present.

But one question remains: Why was ordinary intraword hyphenation still working as normal in the `amsppt` bibliography macros? The answer is that the discretionary hyphens added by TeX to enable automatic hyphenation are not added at the same time as the other breakpoints. All the other kinds of breakpoints are inserted during the initial construction of the horizontal list, but, striving for more efficiency, TeX tries first to make a paragraph without resorting to automatic hyphenation; if and only if this first attempt fails — if line breaks cannot be found such that the badness of each line is less than `\pretolerance` — TeX goes back through the horizontal list of the paragraph and adds the discretionary hyphens indicated by its hyphenation patterns, and goes through another line-breaking pass. On this second pass it also uses `\tolerance` instead of `\pretolerance`.

Thus the `amsppt` bibliography macros first construct the pieces of a paragraph in restricted horizontal mode, so that no breakpoints are added; then the pieces are combined into one long horizontal list and sent to TeX for paragraphing; if the first attempt at paragraphing fails, TeX follows its usual process of adding discretionary hyphens, and tries again to make a paragraph, whereupon hyphenation works as normal.

Using \vboxes instead of \hboxes

The TEXbook, Appendix D, pp. 398–400, has an example that uses the technique of `\unhbox`ing to construct a paragraph out of many short footnotes. Finding no mention there of hyphenation peculiarities, I wrote to Knuth to suggest that a footnote about hyphenation might be useful to add in some future printing, and to ask if there was any way to provide normal line breaking after hyphens in unhboxed text; I couldn't think of any solution short of catcoding the hyphen to be active and having it do some laborious checking to handle the possibility of en-dashes and em-dashes (the need to consider `\relpenalty` and `\binoppenalty` hadn't even occurred to me). In response Knuth outlined an interesting alternative: If instead of an `\hbox` you use a `\vbox` with `\hsize` set to `\maxdimen`, the product will be a one-line paragraph, with all the necessary breakpoints present (because unrestricted, rather than restricted, horizontal mode will be used to construct the horizontal list). There is an extra level of boxing present, but an extra unpacking step will take care of that.

Here is a sketch of the TEXnical details, using a simplified bibliography scheme with three tags: reference label `\key`, author name `\by`, and article title `\paper`.

To start with, some box names need to be declared:

```
\newbox\keybox \newbox\bybox
\newbox\paperbox
```

At the very beginning of a reference, we need to provide a `\bgroup` to match the first upcoming `\egroup`. We do this by setting a `\vbox` that will simply be discarded.

```
\def\ref{\par\setbox0=\vbox\bgroup}
```

And here's the definition of `\key`:

```
\def\key{\par\egroup
 \setbox\keybox=\vbox\bgroup
 \hsize=\maxdimen \noindent\bf}
```

Without the `\noindent` we'd get a box of width `\parindent` because we're beginning a paragraph; this would interfere later when the pieces of the reference are combined.

Actually, since the macros `\by` and `\paper` are nearly identical, it's better to write a generalized macro that can be shared by all three:

```
\def\makerefbox#1#2{\par\egroup
 \setbox#1=\vbox\bgroup
 \hsize=\maxdimen \noindent#2}
```

Then the definitions are

```
\def\key{\makerefbox\keybox\bf}
\def\by{\makerefbox\bybox\rm}
\def\paper{\makerefbox\paperbox\it}
```

`\endref` performs the usual sequence `\par\egroup` to close the final data box, whatever it may be, and then unpacks `\keybox`, `\bybox`, and `\paperbox`, inserting punctuation and space as desired. Since each unpacking operation is the same, it's best done as a macro, say `\unvxh` ("unvbox, extract the last line, and unhbox it").

```
\def\endref{\par\egroup
% preliminary formatting
  \noindent\hangindent\parindent
% reference label
  {\bf[\unvxh\keybox]}\enspace
% author name(s)
  \unvxh\bybox,\space
% article title
  \unvxh\paperbox.\par
}
```

The `\bf` here is necessary if we want bold `[ ]` around the reference key. The contents of `\keybox` are already typeset, so we could not change them to bold at this point if they were not bold already.

The last line of a paragraph ends with three special items:

```
\penalty 10000
\glue(\parfillskip) 0.0 plus 1.0fil
\glue(\rightskip) 0.0
```

If we made sure `\parfillskip` and `\rightskip` are zero, by setting them to zero at the same time as we set `\hsize` to `\maxdimen`, these items could perhaps be left in place. On the other hand, if we remove them, the reassembled reference will more closely resemble a paragraph typeset naturally, and furthermore, it will use slightly less of TEX's main memory. So we remove them using `\unskip` and `\unpenalty` in the macro `\unvxh`.

```
\def\unvxh#1{%
  \setbox0=\vbox{\unvbox#1%
    \global\setbox1=\lastbox}%
  \unhbox1
% remove \rightskip, \parfillskip,
% and penalty
  \unskip\unskip\unpenalty
}
```

Now to try these macros out:

```
\ref \paper Title of the important
   work he wrote
\by Arthur Aja Desc
\key De \endref

\ref\key K\by Kustim Kunsla
\paper And to test line breaking
  after explicit hyphens: pneu-mono-ul-%
  tra-mi-cro-scop-ic-sil-i-co-%
  vol-ca-no-co-ni-o-sis \endref
```

[De] Arthur Aja Desc, *Title of the important work he wrote.*

[K] Kustim Kunsla, *And to test line breaking after explicit hyphens: pneu-mono-ul-tra-mi-cro-scop-ic-sil-i-co-vol-ca-no-co-ni-o-sis.*

Without the use of Knuth's idea there would be no legal breakpoints after any of the explicit hyphens in pneu-mono-ul-tra-mi-cro-scop-ic-sil-i-co-vol-ca-no-co-ni-o-sis and we'd have an overfull line.

Complications

A significant stumbling block was pointed out to me by Ron Whitney at TUG when I submitted this article (thank you, Ron): if we're typesetting a piece of a reference using `\vbox` instead of `\hbox`, explicit line breaks typed by the user will take effect as soon as a `\par` is read — that is, when the information is stored, rather than when it is combined with the rest of the reference. In addition to the `underfull \hbox` message that will result (because `\hsize = \maxdimen`), this means that the `\vbox` will contain more than one line of text, and unpacking it will not be so simple after all.

Let's suppose that, as in $\mathcal{A}\mathcal{M}\mathcal{S}$-TeX, the user has a single command for forcing a line break, called `\linebreak`. We don't need to worry about "suggested" line breaks — anything with a `\penalty` greater than −10000 — because these will remain inactive during the initial typesetting, thanks to the large `\hsize`. The problem is to take the text that is split by a `\linebreak` and save it in such a form that it can later be joined seamlessly with the rest of the reference, but with the line break preserved. And we want to suppress the `underfull \hbox` message while we're at it.

There are various alternatives, and some readers may be able to devise a better solution than the one I chose. But first let me mention briefly a couple of the more tempting and less practical alternatives that I considered:

(1) Tell the users that they can only use pure bibliographic information inside the reference macros, unsullied by uncouth raw typesetting commands like `\linebreak`. This would mean only that users would grumble about their output and line breaking problems would be deferred to the attention of publishers' production troubleshooters, e.g., me.

(2) Redefine `\penalty` to check the penalty amount and make sure it's −9999 or greater; that is, convert forced line breaks to "emphatically suggested" line breaks. This would work reasonably well in a bibliography context (especially with a high setting of `\tolerance`), since penalties are only used for line breaking and page breaking, and within the scope of the `\penalty` redefinition there wouldn't be any embedded vboxes wherein line-breaking had to be restored to normal. However, this alternative seems dangerous; I was able to imagine at least one scenario (too complicated to be worth describing here) where changing forced breaks to nonforced breaks would cause a problem.

One way of handling line breaks. Assume that, as in $\mathcal{A}\mathcal{M}\mathcal{S}$-TeX, the user has a single command, `\linebreak`, to force a line break, and that its normal definition is essentially `\penalty-10000` (ignoring some frills like error messages if in vertical mode). We don't have to worry about penalties greater than −10000, as mentioned earlier. Inside the `\vbox` that is being typeset by `\makerefbox`, we can change the definition of `\linebreak`:

```
\def\linebreak{\par
  \setbox0=\lastbox
  \setbox\holdoverbox=
    \hbox{\unhbox\holdoverbox
```

(in case more than one `\linebreak` occurs within a single piece of the reference)

```
      \unhbox0
      \unskip\unskip\unpenalty
      \penalty-10000}%
  \noindent}

% Can't forget this
\newbox\holdoverbox
```

Thus `\linebreak` takes the text so far and saves it in `\holdoverbox`, along with a break penalty, inactive here because we're in an `\hbox` instead of a `\vbox`. This saved part will then be combined with the remainder of the current reference field, by way of some extra processing in `\makerefbox` and `\endref`.

```
\def\makerefbox#1#2{\par
  \checkholdoverbox \egroup
  \setbox#1=\vbox\bgroup
  \hsize=\maxdimen \noindent#2}
```

```
\def\endref{\par
  \checkholdoverbox\egroup
  \noindent\hangindent\parindent
  {\bf [\unvxh\keybox]}\enspace
  \unvxh\bybox,\space
  \unvxh\paperbox.\par}

\def\checkholdoverbox{%
 \ifvoid\holdoverbox
  \else \setbox0=\lastbox
 \hbox{\unhbox\holdoverbox
   \unhbox0}\fi}
```

After `\checkholdoverbox`, we have reduced the contents of the current `\vbox` to a single `\hbox`, just as if no `\linebreak` had been present.

There are extra complications if the user inserts a `\linebreak` at the end of a field, because that means the break will be taken between the text and ensuing automatic punctuation if we don't do something about it. Some nice checking and rearranging to handle this case was present in Spivak's original version of the amsppt reference macros and will not be discussed here.

Text measurement applications

To understand more clearly the second application mentioned at this article's beginning, consider the usual method for measuring a piece of text and using the width as a selector:

```
\def\caption#1{%
  \setbox0=\hbox{#1}%
  \ifdim\wd0<\hsize
   \centerline{\box0}% centered line
  \else
   \noindent#1\par % paragraph
  \fi}
```

Using `\unhbox0` in the ⟨*paragraph*⟩ branch would be slightly more efficient — it avoids typesetting the text twice. But that would bring in the line-breaking problems described above. We can have our cake and speed up, too, if we use Knuth's idea and start by setting the caption in a `\vbox`, rather than an `\hbox`.

Handling vertical mode material in a caption. After setting material in a `\vbox`, simply extracting `\lastbox` as in the `\unvxh` example may not be enough. What if someone wants two paragraphs in a caption, or maybe even a displayed equation? A `\par` or display in the `\vbox` will mean that after extracting `\lastbox` some material will be left behind. I had a chance to experiment in a recent assignment, which was to create a LaTeX documentstyle for electronic submissions to American Mathematical Society journals. One of the macros I had to modify was `\@makecaption`, whose original definition from `article.sty` was roughly the same as the `\caption` example above.

This is what `\@makecaption` had to be modified to do: If the total width of the caption material is greater than `\columnwidth` (29pc), break the caption into lines using a line width of 23pc, and center the resulting block between the margins. Otherwise set the caption as a single line, centered between the margins.

To do this I decided to set the caption as a `\vbox` with line width 23pc, but allow the last line (which may be the only line) to be up to 29pc long by adding a kern of –6pc. The last line is put into box register 1 using `\lastbox`. After extracting the last line, if there is anything left in the `\vbox`, that means the caption was (most likely) more than one line long and some extra processing is needed.

The first argument of `\@makecaption` is the name of the figure or table, e.g., "Figure 1", generated automatically by LaTeX. The second argument is the caption text typed by the user.

```
\long\def\@makecaption#1#2{%
```

We begin by setting the text in a `\vbox`.

```
  \setbox\@tempboxa\vbox{%
% hsize := 29 - 6 = 23 pc
   \advance\hsize-6pc\noindent
```

Ordinarily the `\unskip` here would be done automatically by `\par`, but here the `\kern` gets in the way so we must do the `\unskip` explicitly.

```
   {\sc#1}\enspace#2\unskip
      \kern-6pc\par
   \global\setbox\@ne\lastbox}%
```

Now box 1 holds either the entire caption, or the last line of a multiline caption. In either case we want to remove `\parfillskip`, `\rightskip`, and the `\kern` of –6pc. Before we get to the `\kern` we also have to remove the penalty of 10000 that is inserted by TeX at the end of every paragraph.

```
  \setbox\@ne\hbox{\unhbox\@ne
    \unskip\unskip\unpenalty\unkern}%
```

If `\@tempboxa` is not empty at this point it means the caption was more than one line long. In that case we reset the caption using the contents of `\@tempboxa` and `\unhbox`ing box 1 (because the contents of box 1 may need to be made into two lines instead of one, if its length is greater than 23 picas). Otherwise the caption material is made into a single centered line. Note: A box register containing an empty box is not the same as a void box register; a box register that contains `\vbox{}` will not return true if tested with the `\ifvoid` test. So to decide whether `\@tempboxa` is empty we cannot use `\ifvoid`. Instead we employ the simple strategy of measuring the width of the box. This will not be 100% failsafe but the failure cases that I've been able to imagine are all rather exotic.

```
\ifdim\wd\@tempboxa=\z@
  \setbox\@ne\hbox to\columnwidth{%
     \hss\kern-6pc\box\@ne\hss}%
\else % more than one line
   \setbox\@ne\vbox{\unvbox\@tempboxa
      \noindent\unhbox\@ne
      \advance\hsize-6pc\par}%
\fi
```

The `\kern-6pc` in the first branch is to offset the `\moveright` that is about to be done next. (If tortured, I would be forced to admit that it took me several attempts before I figured out the right amount for this kern and the proper place to put it.) Finally, we put the caption on the page, with a `\vskip` to separate it from the preceding or following material.

```
  \ifnum\@tempcnta<64 %if it's a figure
    \vskip 1pc%
    \moveright 3pc\box\@ne
  \else % if the float IS NOT a figure
    \moveright 3pc\box\@ne
    \vskip 1pc%
  \fi
}
```

By testing `\@tempcnta` we can tell whether the caption is being used in a `figure` environment or not; if so, we assume that the caption is placed below the artwork and hence put the `\vskip` above the caption; otherwise we assume the caption is at the top of the floating insertion and we put the `\vskip` below it.

`\@makecaption` presents a few extra complications that have been omitted for the sake of simplicity; as given here, the caption will not be quite centered if the figure caption has no text, and so on.

◇ Michael Downes
American Mathematical Society
201 Charles Street
Providence, RI 02904
mjd@Math.AMS.com

Looking Ahead for a ⟨box⟩

Sonja Maus

TeX's primitive `\afterassignment` can be used for macros which first assign a value to a parameter, and then perform some actions using that value. For instance the plain TeX macros `\magnification` and `\hglue` (see *The TeXbook*, p. 364 and 352), assign a ⟨number⟩ or ⟨glue⟩ value to a variable and then use this value. They provide a user-friendly "syntax mimicry": `\magnification` looks like an integer parameter in an assignment, and `\hglue` looks like the primitive command `\hskip`. There is another advantage to this method over the use of arguments with `#1`: At the moment when TeX looks at the tokens of the value, it already knows what kind of value it is looking for. This would be very useful when the value to be read is a ⟨box⟩, because an explicit `\hbox` or `\vbox` may contain `\catcode` changes and all tokens should not be read ahead.

There are seven ways to write a ⟨box⟩ (*The TeXbook*, p. 278). The `\afterassignment` command behaves differently with the first four and the last three of these ⟨box⟩es:

```
\afterassignment\t \setbox0=\box1
```

results in `\setbox0=\box1 \t`, whereas

```
\afterassignment\t \setbox0=\hbox{h}
```

results in `\setbox0=\hbox{\t h}`.

The macro `\afterbox` gives a substitute which is equally valid for all ⟨box⟩es. Its syntax is

```
\afterbox<argument><box>
```

where ⟨argument⟩ is an argument for an undelimited macro parameter (see *The TeXbook*, p. 204), i.e. a single token or several tokens in explicit braces. `\afterbox` puts the ⟨argument⟩ aside (without the braces, if any), assigns the ⟨box⟩ to the register `\box\afbox`, and then reads the ⟨argument⟩ again.

The definition must be read when `@` is a letter:

```
\newbox\afbox
\def\afterbox#1{\def\afb@xarg{#1}%
  \afterassignment\afb@x
  \chardef\next`.}
\def\afb@x{\futurelet\next\afb@xtest}
\def\afb@xtest
 {\ifcase\ifx\next\hbox\tw@\fi
         \ifx\next\vbox\tw@\fi
         \ifx\next\vtop\tw@\fi
         \ifx\next\box\@ne\fi
         \ifx\next\copy\@ne\fi
         \ifx\next\vsplit\@ne\fi
         \ifx\next\lastbox\@ne\fi
         0\errmessage{No <box>}%
  \or\afterassignment\afb@xarg
```

```
    \or\afterassignment\afb@xagarg
    \fi
    \setbox\afbox}
  \def\afb@xagarg{\aftergroup\afb@xarg}
```

First, `\afterbox` puts the ⟨argument⟩ into `\afb@xarg`. Then the `\chardef` command reads a ⟨number⟩ which turns out to be a ⟨normal integer⟩ with a ⟨character token⟩ (see *The TeXbook*, p. 269). As the syntax of ⟨number⟩ requires, TeX expands tokens and looks for ⟨one optional space⟩ which turns out ⟨empty⟩. This looks crazy, but it has the effect of unpacking the first non-expandable token of ⟨box⟩ if it was hidden behind expandable tokens like `\null` or `\line` (or `\Boxit` below). This non-expandable token's meaning is then assigned to `\next` and tested by `\afb@xtest`. It must be one of the seven primitives listed with the `\ifx`s, and the cases 1 and 2 correspond to the two behaviours of `\afterassignment` mentioned above. In both cases, `\afb@xarg` will reappear exactly at the time when the `\setbox` assignment is finished, e.g.:

```
\afterbox \t \box1
```

results in `\setbox\afbox=\box1 \t`, whereas

```
\afterbox \t \hbox{h}
```

first becomes `...\hbox{\afb@xagarg h}` and then results in `\setbox\afbox=\hbox{h}\t`.

For example,

```
\def\Boxit{\hbox\bgroup\afterbox
   {\vrule
    \dimen0=\dp\afbox
    \advance\dimen0 by3.4pt
    \lower\dimen0 \vbox
     {\hrule \kern3pt
      \hbox{\kern3pt\box\afbox\kern3pt}
      \kern3pt \hrule}%
    \vrule \egroup}}
```

solves Ex. 21.3 of *The TeXbook* with `\Boxit<box>` instead of `\boxit{<box>}`, and `\Boxit<box>` is itself a ⟨box⟩, so that `\Boxit\Boxit<box>` makes a double frame. The macro `\framedhbox` defined by

```
\def\framedhbox{\Boxit\hbox}
```

can be used exactly like the primitive `\hbox`:

```
\framedhbox{<horizontal material>}
```

It can also be `\raise`d, or assigned to a box register, and `to` or `spread` can be specified.

◇ Sonja Maus
Memelweg 2
5300 Bonn 1
Federal Republic of Germany

An Indentation Scheme

Victor Eijkhout

Indentation is one of the simpler things in TeX: if you leave one input line open you get a new paragraph, and it is indented unless you say `\noindent`. And if you get tired of writing `\noindent` all of the time, you declare

```
\parindent=0pt
```

at the start of your document. Easy.

More sophisticated approaches to indentation are possible, however. In this article I will sketch a quite general approach that can easily be incorporated in existing macro packages. For a better appreciation of what goes on, I will start with a tutorial section on what happens when TeX starts a paragraph.

1 Tutorial: paragraph start

When TeX is not busy typesetting mathematics, it is processing in *horizontal mode*, or *vertical mode*. In horizontal mode it is putting objects — usually characters — next to each other; in vertical mode it is putting objects — usually lines of text — on top of each other.

To see that there is a difference, run the following pieces of code through TeX:

```
\hbox{a}
\hbox{b}
\bye
```

and

```
a
\hbox{b}
\hbox{c}
\bye
```

You notice that the same objects are treated in two different ways. The reason for this is that TeX starts each job in vertical mode, that is, stacking material. In the second piece of input TeX saw the character 'a' before it saw the boxes. A character is for TeX the sign to switch to horizontal mode, that is, lining up material, and start building a paragraph.

Commands that can make TeX switch to horizontal mode are called 'horizontal commands'. As appeared from the above two examples characters are horizontal commands, but boxes are not. Let us now look at the two most obvious horizontal commands: `\indent` and `\noindent`.

1.1 `\indent` and `\noindent`

`\indent` is the command to start a paragraph with indentation. TeX realizes the indentation by inserting a box of width `\parindent`. If you say `\indent` somewhere in the middle of a paragraph you get some white space there, caused by the empty box.

`\noindent` is the command to start a paragraph without indentation. After this command TEX merely switches to horizontal mode; no indentation box is inserted. If you give this command somewhere in the middle of a paragraph it has no effect at all.

If TEX sees a horizontal command that is not `\indent` or `\noindent`, for instance a character, it acts as if the command was preceded by `\indent`. This is why paragraphs usually start with an indentation.

As an illustration here is a small variation on the above two examples:

```
\noindent
\hbox{a}
\hbox{b}

\indent
\hbox{a}
\hbox{b}
\bye
```

Now in both cases the boxes are part of a paragraph that was explicitly begun with `\indent` or `\noindent`.

1.2 Using `\everypar`

TEX performs another action when it starts a paragraph: it inserts whatever is currently the contents of the token list `\everypar`. Usually you don't notice this, because the token list is empty in plain TEX (the TEX book [1] gives only a simple example, and the exhortation 'if you let your imagination run you will think of better applications'). LATEX [2], however, makes regular use of `\everypar`. Some mega-trickery with `\everypar` can be found in [3].

• Just to show how this works, I put in front of this paragraph the statement

```
\everypar={$\bullet\quad$}
```

That is, I told TEX that `$\bullet\quad$` should be inserted in front of a paragraph.

• There's nothing specified for this paragraph; I get the bullet for free, as `\everypar` does exactly what its name promises: it is inserted in front of *every* paragraph.

At the end of the previous paragraph I specified

```
\everypar={}
```

so nothing is inserted from this paragraph onwards.

1.3 Removing indentation

Every TEX user knows that indentation can be prevented globally by setting `\parindent` to zero. However, this is rather crude, and if you use the plain TEX macros you may notice several rather unpleasant side effects of this action, for instance when you use the macros `\item` and `\footnote`.

It is possible to use `\everypar` to prevent indentation, or more correctly: to remove indentation. This can be achieved by

```
\everypar={{\setbox0=\lastbox}}
```

This needs some explanation.

If the last item that was processed by TEX is a box, then that box is accessible by the command `\lastbox`. If the last item was not a box then `\lastbox` is an empty box, but no error ensues. As the `\everypar` list is inserted after any indentation box, the `\lastbox` command will get hold of the indentation box if there is one. By assigning the last box to another box register — here `\box0` — it is removed from where it was previously.

Finally, the statement

```
\setbox0=\lastbox
```

is enclosed in braces. TEX's grouping mechanism restores values when the group ends that were current when the group began. In this case it has the effect of totally removing the indentation box: first it is taken and assigned to `\box0`, then the value of `\box0` is restored to whatever it was before the group began.

1.4 Other actions at the start of a paragraph

In the above discussion I have omitted one action that takes place at the start of a paragraph: TEX inserts (vertical) `\parskip` glue above the paragraph. As this has no relevance for the subject of indentation I will not go into it any further. However, in a subsequent article I will give more information about `\parskip`.

2 To indent or not to indent

In classical book typography [4] every paragraph is indented, with the exception of the first paragraph of a chapter or section. Nowadays a design where no paragraph indents is quite common. There are two mixtures between always indenting and never indenting: occasionally indenting, and occasionally not indenting. Thus it seems possible to characterize indentation strategies by two yes/no parameters: one that decides whether paragraphs should indent in principle, and another parameter that can over-rule those decisions. Let us now see how this can be implemented in TeX.

2.1 Implementation

Above I have already indicated that changes to `\parindent` should be avoided. Let us then assume that `\parindent` is greater than zero, even if we will never indent a paragraph (see [5] for other uses for the `\parindent` quantity). We must then realize unindented paragraphs by removing their indentation as explained above.

First we need a macro for removing the indentation:

```
\def\removeindentation
    {{\setbox0=\lastbox}}
```

Then we need the switches that control indentation:

```
\newif\ifNeedIndent %as a rule
\newif\ifneedindent %special cases
```

Now for the definition of `\everypar`. This is a bit tricky.

Let us first collect some bits and pieces. The main question is to decide when `\removeindent` should be called. This is for instance the case if `\NeedIndentfalse`, and that parameter is not overruled by `\needindenttrue`.

```
\ifNeedIndent
    \ifneedindent
    \else \removeindentation
\fi \fi
```

Indentation should also be removed when `\needindentfalse` overrules the general parameter `\NeedIndenttrue`.

```
\ifNeedIndent
\else \ifneedindent
      \else \removeindentation
\fi   \fi
```

Next we should make sure that `\ifneedindent` is used only for exceptional cases: if the user or a macro sets this parameter to a different value from `\ifNeedIndent`, then that should be obeyed exactly once.

```
\ifNeedIndent
      \ifneedindent
      \else \needindenttrue \fi
\else \ifneedindent \needindentfalse
\fi   \fi
```

This is then the full definition of `\everypar`:

```
\everypar={\controlledindentation}
\def\controlledindentation
   {\ifNeedIndent
          \ifneedindent
          \else \removeindentation
                \needindenttrue
          \fi
    \else \ifneedindent
                \needindentfalse
          \else \removeindentation
    \fi   \fi}
```

Another implementation would be possible:

```
\def\controlledindentation
   {\ifneedindent
    \else \removeindentation \fi
    \let\ifneedindent=\ifNeedIndent}
```

This saves one conditional, but for most paragraphs it involves an unnecessary `\let` command.

2.2 Usage

My aim in developing this indentation scheme was to hide all commands pertaining to indentation in macros. The user should have to specify only once whether paragraphs should indent as a rule:

```
\NeedIndenttrue
```

and then macros should declare the exceptions:

```
\def\section#1{...
    \needindentfalse
    ...}
```

2.3 But couldn't you simply ...?

Maybe people who read this have written macros themselves that end like

```
\def\section#1{...
    ...
    \noindent}
```

or

```
\def\section#1\par{...
    ...
    \noindent}
```

This works reasonably well, but it is not completely safe. In the first case there shouldn't be an empty line after a

```
\section{...}
```

call, and in the second case there can only be one empty line after

```
\section ...
```

The reason for this is that *every* empty line generates a `\par` command, which annuls the effect of the `\noindent`. Hence the more drastic approach.

An argument the other way around can also be found, by the way. As Ron Whitney pointed out to me, the following piece of code causes trouble:

```
\section{Title}
{\smallcaps The first} words are ...
```

Any changes made by `\everypar` are now effected *inside a group*. In this case one remedy is to insert a `\leavevmode` command, or to define

```
\def\smallcapswords#1{\leavevmode
   {\smallcaps #1}}
```

which can be used at any place.

Another remedy would be to let all assignments controlling indentation be global. However, there are some subtle objections to this.

3 About macro packages and users

Above I remarked that plain TEX does not use `\everypar`, and that LATEX redefines it a lot. This means that in plain TEX the user is free to take every value of `\everypar` that he or she likes; in LATEX every attempt of the user to use `\everypar` is immediately thwarted.

One might ask how the use of `\everypar` that I have sketched compares to this. Can the user be allowed to access `\everypar`, even if the macro package needs it all the time?

In my own 'Lollipop' format I have taken the following way out. The user or the style designer is allowed to fill in `\everypar`, as long as the statement

```
\the\everyeverypar
```

is included. Here `\everyeverypar` is the token list with the constant actions such as indentation control that should be performed always.

A format designer who wishes to hide even this from the user or the style designer, could use the following piece of code

```
\newtoks\temppar
\def\everyparagraph
   {\afterassignment\xevpar
    \temppar}
\def\xevpar
   {\edef\act{\everypar=
         {\the\everyeverypar
          \the\temppar
         }}%
    \act}
```

so that it becomes possible to write

```
\everyparagraph={\DoSomething
     \everyparagraph={}}
```

while the `\everypar` will still contain all of the constant actions.

Short explanation: `\everyparagraph` is a macro that is made to look like a token paramter by the use of `\afterassignment`. This latter command sets aside `\xevpar` for execution after whatever follows is assigned to `\temppar`. Following the assignment, the macro `\xevpar` unwraps the `\temppar` token list and the constant actions into `\everypar`.

4 Conclusion

In a systematic layout indentation commands need never be typed by the user; they can all be hidden in macros. Using `\everypar` it is possible to prevent indentation both in single instances, and throughout the document. This has the advantage that is is not necessary to zero the `\parindent` parameter or use `\indent` and `\noindent` instructions.

The approach of employing `\everypar` as sketched above can also be used for a paragraph skip scheme, as I will show in the subsequent article.

References

[1] Donald Knuth, The TEXbook, Addison-Wesley Publishing Company, 1984.

[2] Leslie Lamport, LATEX, a document preparation system, Addison-Wesley Publishing Company, 1986.

[3] Victor Eijkhout, Unusual paragraph shapes, *TUGboat* vol. 11 (1990) #1, pp. 51–53.

[4] Stanley Morison, First principles of typography, Cambridge University Press, 1936.

[5] J. Braams, V. Eijkhout, N.A.F.M. Poppelier, The development of national LATEX styles, *TUGboat* vol. 10 (1989) #3, pp. 401–406.

A `\parskip` Scheme

Victor Eijkhout

While I was working on the LATEX styles described in [1], it became apparent to me that lots of people are rather fond of the sort of layout that can be described as

```
\parindent=0cm
\parskip=6pt % or other positive size
```

Unfortunately, most of them realize this layout by no more sophisticated means than simply inserting these two lines at the beginning of the input. The drawback of such a simple action is that all sorts of vertical spaces are augmented by the `\parskip` when there is absolutely no need to, or where it is positively unwanted. Examples of this are the white space below section headings, and the white space above and below list environments in LATEX.

In this article I will present an approach that unifies the paragraph skip and the white spaces surrounding various environments. Since the macros given below make use of the `\everypar` token list, this article may be seen as a sequel to the previous article in this issue of *TUGboat* concerning an indentation scheme, which is based on a similar principle. The `\everypar` parameter was explained there.

1 `\parskip`

TeX starts a paragraph when it switches from vertical to horizontal mode. The vertical mode may have been initiated by a `\par` (for instance because of an empty line after a preceding paragraph) or by a vertical skip command; the switch to horizontal mode can be effected by, for example, a character or a horizontal skip command (see the list in [2, p. 283]). Immediately above the first line of the paragraph TeX will then add glue of size `\parskip` to the vertical list[1].

Apparently, then, the `\parskip` parameter is very simple to use. That this is only an apparent simplicity becomes clear in a number of instances.

For instance, unless precautions are taken, the white space below headings is augmented by the paragraph skip. Precautions against this are not particularly elegant: the easiest solution is to include a

```
\vskip-\parskip
```

statement, to backspace the paragraph skip in advance. Such an approach, however, is somewhat error-prone. Vertical spacing will be messed up if what follows is not a paragraph, but a display formula or a box.

Similar considerations apply to the amounts of white space that surround, for example, list environments, as in LaTeX.

2 Paragraph skip: to be or not to be

(This section is something of a footnote to the rest of the article. Readers who are not interested in layout considerations may skip the rest of it.)

Ordinarily in plain TeX and in LaTeX the paragraph skip is set to `0pt plus 1pt`, which gives pages some 'last resort' stretchability. However, even an amount of vertical space as small as one point may become very visible, and often without need (see for instance the first page of the preface of [2]).

Furthermore, Stanley Morison states that not indenting paragraphs is 'decidedly an abject method' [3]. However, reading his intention instead of his words, he is only concerned with the recognizability of the individual paragraphs. The positive value of the paragraph skip is sufficient to ensure this. If a layout is based on zero values for both `\parindent` and `\parskip`, one may for instance give the `\parfillskip` a positive natural width to prevent last lines of a paragraph from almost, or completely, lining up with the right margin.

Neither Donald Knuth nor Leslie Lamport seems to have given much thought to the case where the paragraph skip has a positive natural width. Leslie Lamport dismisses all potential difficulties with the remark that 'it is customary not to leave any extra space between paragraphs' [4, p. 94].

3 Environments and white lines

Given that the paragraph skip appears to interact with explicit vertical spacing in user macros, it may seem like a good idea to find a unified approach to both. In the rest of this article I will describe the implementation of the following basic idea: *give the paragraph skip the value zero whenever you do an explicit vertical skip.*

For the presentation I assume a context with some form of environments. These are the assumptions that I make about such environments:

- An environment is a portion of material that is vertically separated from whatever is before and after it. Thus, according to this definition, a portion of a paragraph cannot be an environment, nor can an environment start or end in the middle of a paragraph.
- An environment has associated with it three glue parameters: to an environment `foo` correspond `\fooStartskip` (glue above the environment), `\fooParskip` (the paragraph skip inside the environment), and the `\fooEndskip` (glue below the environment).
- At the outset of the environment a

  ```
  \StartEnvironment{foo}
  ```

  statement is executed; at the end of the environment a macro

  ```
  \EndEnvironment{foo}
  ```

  is executed. These statements are assumed to contain a `\begingroup` and `\endgroup` respectively.

Such assumptions are sufficiently general for the macros below to be adaptable to existing macro packages. At first sight it would appear as if section headings are not covered by the above points. However, there is no argument against the start and end of an environment occurring in the same macro.

[1] Unless this paragraph is at the start of a vertical list, for instance at the start of a vertical box or insertion item.

4 Tools

First I will present two auxiliary macros: `\csarg` and `\vspace`.

The command `\csarg` is only needed inside other macros; it is meant to enable constructs such as

```
\csarg\vskip{#1Parskip}
```

Its definition is

```
\def\csarg#1#2{\expandafter
    #1\csname#2\endcsname}
```

By way of explanation of this macro, consider a simple example. Let us assume that there exists a macro

```
\def\startlist#1{ ...
    \csarg\vskip{#1Startskip}
    ...}
```

The call

```
\startlist{enumerate}
```

will then lead to the following call to `\csarg`:

```
\csarg\vskip{enumerateStartskip}
```

This expands to

```
\expandafter\vskip
 \csname enumerateStartskip\endcsname
```

Now the `\expandafter` forces `\csname` to be executed before the `\vskip`, so the next step of the expansion looks like

```
\vskip\enumerateStartskip
```

and this statement can simply be executed.

Next I need a generalization of `\vskip`, which I will call `\vspace`: *a number of calls to* `\vspace` should have the effect that only the maximum argument is placed.

```
\newskip\tempskipa
\def\vspace #1{%
  \tempskipa=#1\relax
  \ifvmode \ifdim\tempskipa<\lastskip
     \else \vskip-\lastskip \vskip\tempskipa
     \fi
  \else \vskip\tempskipa \fi}
```

This may need some explanation, too. First, by the assignment

```
\tempskipa=#1
```

I allow the argument of `\vspace` to be both a control sequence, for instance `\parskip`, and a denotation, for instance `5pt plus 3pt`. If one omits the assignment, the latter option would cause trouble in the `\ifdim` test.

The decision to keep the maximum value of the skip, instead of always replacing the last skip, was motivated by phenomena such as a display formula at the end of a list. If the skip below the display is larger than the vertical glue below the list (which may for instance be zero), the former should be retained entirely.

Note that this macro will insert its argument even if it has the same size as the last skip. There is a good reason for this. If the call to `\vspace` follows a `\par` command at the end of a paragraph, it is called in vertical mode, but the last item on the vertical list is a box (the last line of the paragraph) instead of a glue item. The parameter `\lastskip` will then be zero. If the argument to `\vspace` is something like `0pt plus 5pt` we still want it to be added to the list, even though its natural size is zero.

5 The environment macros

In this section, I will give the implementation of the macros `\StartEnvironment` and `\EndEnvironment`.

There is a remarkable similarity between these two macros. As I explained above, the basic idea is to have only explicit spacing above and below the environment; thus, the value of `\parskip` should then be zero both for the first paragraph in the environment, and for the first paragraph that follows it. Both macros should then

- save the current value of the paragraph skip;
- set the paragraph skip to zero;
- give TeX a signal that, somewhere in the near future, the old value of `\parskip` is to be restored.

For this I allocate a skip register and a conditional:

```
\newskip\TempParskip
\newif\ifParskipNeedsRestoring
```

The basic sequence for the starting and ending macros is then

```
\TempParskip=\parskip
\parskip=0cm\relax
\ParskipNeedsRestoringtrue
```

For both macros, however, this sequence needs to be refined slightly.

The paragraph skip to be 'restored' at the start of the environment is the specific value associated with that environment. This gives us:

```
\def\StartEnvironment
 #1{\csarg\vspace{#1Startskip}
    \begingroup %% make changes local
    \csarg\TempParskip{#1Parskip}
    \parskip=0cm\relax
    \ParskipNeedsRestoringtrue}
```

Note that the statement

```
\csarg\TempParskip{#1Parskip}
```

denotes an assignment (without an optional equals sign) here.

At the end of the environment we have to be more careful. It may be the case that the environment being ended was inside another environment, and occurred before the first paragraph inside that environment. In that case the value of `\parskip` is zero, and the proper value must still be restored. Therefore, no further actions are required. We arrive at the following implementation:

```
\def\EndEnvironment
 #1{\csarg\vspace{#1Endskip}
    \endgroup %% restore global values
    \ifParskipNeedsRestoring
    \else \TempParskip=\parskip
          \parskip=0cm\relax
          \ParskipNeedsRestoringtrue
    \fi}
```

Note that both macros start with a vertical skip. This prevents the `\begingroup` and `\endgroup` statements from occurring in a paragraph. On a side note: since these macros are executed in vertical mode, I have not bothered to terminate any lines with comment signs. Any spaces generated by these macros are ignored in vertical mode.

6 Paragraph skip restoring

So far, I have ignored one important question: how exactly is restoring the paragraph skip implemented. For this I use the `\everypar` token list. Basically then, the idea is the same as in "An Indentation Scheme" (p. 612): the occurrence of a paragraph will automatically have TEX perform, through the insertion of the `\everypar` tokens, the actions necessary for subsequent paragraphs.

```
\everypar=
   {\ControlledIndentation
      %see ``An Indentation Scheme''
    \ControlledParskip}
\def\ControlledParskip
   {\ifParskipNeedsRestoring
       \parskip=\TempParskip
       \ParskipNeedsRestoringfalse
    \fi}
```

The cost of a controlled paragraph skip is then one conditional per paragraph. Conceivably, this cost could even be reduced further (to almost zero) by defining

```
\def\CPS % Controlled Parskip
   {\ifParskipNeedsRestoring
       \parskip=\TempParskip
       \ParskipNeedsRestoringfalse
       \let\ControlledParskip=\relax
    \fi}
```

and including a statement

```
\let\ControlledParskip=\CPS
```

in both the `\StartEnvironment` and `\EndEnvironment` macros, and at the start of the job (for instance by including it in the macro package). This approach, however, does not particularly appeal to me. Too much 'pushing the bit around'.

7 Conclusion

The `\parskip` parameter is arguably the most tricky parameter of TEX. Its workings are very easy to describe, but in actual practice difficulties arise. In this article I have described how treatment of the paragraph skip can be integrated with the glue above and below environments. As in an earlier article on indentation, I use for this the `\everypar` parameter as an essential tool.

References

[1] J. Braams, V. Eijkhout, N.A.F.M. Poppelier, The development of national LATEX styles, *TUGboat* vol. 10 (1989) #3, pp. 401–406.

[2] Donald Knuth, The TEXbook, Addison-Wesley Publishing Company, 1984.

[3] Stanley Morison, First principles of typography, Cambridge University Press, 1936.

[4] Leslie Lamport, LATEX, a document preparation system, Addison-Wesley Publishing Company, 1986.

◇ Victor Eijkhout
Center for Supercomputing
Research and Development
University of Illinois
305 Talbot Laboratory
104 South Wright Street
Urbana, Illinois 61801-2932, USA
eijkhout@csrd.uiuc.edu

Sanitizing Control Sequences Under \write

Ron Whitney

TEX's `\write` primitive is typically used to send information to other files for typesetting later. The most obvious examples here are those cases where chapter heads, section heads, and keywords are written to files along with corresponding page numbers to make tables of contents and indexes. (See Salomon in *TUGboat* 10, no. 3; see also Durst in the same issue for another use of `\write`)

The syntax of a `\write` statement is

`\write`⟨*number*⟩`{`⟨*token string*⟩`}`

where ⟨*number*⟩ is a "stream number" usually allocated by the `\newwrite` macro and corresponds to a disk file opened by `\openout`. To make matters concrete here, we will simply assume that the user has called `\newwrite\outfile` and that our syntax is

`\write\outfile{`⟨*token string*⟩`}`

The execution of the `\write` statement then either involves writing the ⟨*token string*⟩ directly (i.e. `\immediate`ly) to the file corresponding to `\outfile`, or writing the ⟨*token string*⟩ to a node placed in TEX's main vertical list. In the latter case, the ⟨*token list*⟩ in the node is later written to the file corresponding to `\outfile` as the page on which the node is placed undergoes `\shipout` to the `dvi` file.

In either case, as the ⟨*token list*⟩ is transferred to the file `\outfile`, it is fully expanded (that is, the ⟨*token list*⟩ is expanded as, say, it would be under an `\edef`; unexpandable control sequences are written out using the `\escapechar` and the letters or symbols which go to make up their names). There's Good News and Bad News in this. The Good News is that the information to be written to external files is often 'contained' within control sequences, and we definitely want tokens such as `\number` to be expanded (otherwise an index might show every item appearing on, literally, `\number\pageno`). The Bad News is that we would really much prefer that indexes contain items like `\cos` rather than their expansions (e.g. `\mathop{\rm cos}\nolimits`). The Bad News isn't really all that Bad, though, since we can use TEX's `\noexpand` and say

`\write\outfile{\noexpand\cos}`

to achieve what we want.

Then: Where's the Beef? The News isn't really Totally Good because our solution requires some knowledge of the contexts in which `\noexpand` is appropriate. `\write` statements are typically hidden a few levels down inside macros and one might ever know (barring authorship or clear documentation of the underlying macros) when silliness such as `\noexpand` is required. Lamport has done an admirable job in schooling LATEX users to use `\protect` for this very purpose, but even so, questions such as Chris Hand's (*TUGboat* 11, no. 3, p. 456) are natural. In Chris' case, a guillement (`\<<`) caused a section head to expand to some 509 characters in an `.aux` file, and that line was too long for TEX to handle when the `.aux` file was reread. It is also common to see control sequences for accents make for inscrutable tables of contents files.

A Better Method

Much nicer than user-keyboarded `\noexpand`s would be some method of preventing expansion within macros themselves which use `\write`, thus not placing a user under the strain of being on the lookout for expanding arcana. This note proposes a way to handle things generally. It was suggested by a technique used by Michael Wichura in *TUGboat* 11, no. 1 along with simultaneous consideration of Peter Breitenlohner's piece on avoiding long records in `\write` streams in the same issue of *TUGboat*. Other people have undoubtedly thought of the same thing (see *The TEXbook*, p. 382).

The TEX primitive `\meaning` disgorges a "meaning" of whatever token follows it. In the case of a defined control sequence (and let's assume this control sequence has no parameters), say `\foo`, `\meaning\foo` will cause TEX to spit up the sequence `macro:->` followed by a sequence of character tokens as would be obtained by `\string`ing the tokens of `\foo`'s definition. The definition is thus shown as a string of character tokens, all of category 12 (except spaces).

For example, if `\foo` is defined to be the token string `$\sin$ and $\cos$`, its definition consists of 11 tokens altogether: 4 math shifts, 2 spaces, 3 letters, and 2 defined (in `plain`) control sequences. On the other hand, `\meaning\foo` produces the string

```
macro:->$\sin $ and $\cos $
```

whose right part (beyond the `:->`) consists of 4 space tokens and 15 character tokens of category 12. None of this material is expandable; internal objects which had been single tokens (such as `\sin`) are now divided into character tokens (such as `\-s-i-n`).

So, in order to suppress expansion in a ⟨*token string*⟩ which is to be written out to an external file, one need only stuff the ⟨*token string*⟩ into a macro, regurgitate the macro's `\meaning` into another macro, and `\write` the second macro out. The data being written out is, in a certain sense, inert because control sequences have been divided into the characters forming their names and there is nothing to be expanded. If this information is reread from an external file again, however, (and therefore passes through TeX's mouth again) it can be reassembled into the original ⟨*token string*⟩.

To this end, we make the following definition:

```
\def\GetMacroMeaning#1:->#2:->#3\endget{%
  \def#3{#2}}
```

The third argument to `\GetMacroMeaning` is the control sequence into which the "meaning" will be placed. The first two arguments will be produced by 'expanding' `\meaning`. Thus, converting text for, say, a section head which a user keys as `\section{...}`, can be accomplished by inserting the following sort of code within the definition of `\section`:

```
    \def\section #1{%
      ...
      \def\sectionhead{#1}%
      \expandafter\GetMacroMeaning
        \meaning\sectionhead
        :->\xxsectionhead\endget
      \write\outfile
        \expandafter{\xxsectionhead}%
      ...
      }
```

The first `\expandafter` causes `\meaning` to gobble `\sectionhead` and expand into the first 2 arguments of `\GetMacroMeaning`. The second `\expandafter` is used in the `\write` statement because the code presented always stores the head to be written out in `\xxsectionhead`. When 2 section heads occur on the same page, the second will overwrite the first definition of `\xxsectionhead`, so we must make sure that the contents of `\xxsectionhead` gets placed in the node on the page and not just the token `\xxsectionhead` itself. In situations where one may write `\immediate`ly, the line under discussion could become

```
\immediate\write\outfile{\xxsectionhead}%
```

Further Problems

The method above concerns itself only with material we wish to block from expansion as we `\write` it out. As noted previously, other data (such as page and section numbers) must be expanded to get tables of contents and indexes right. David Salomon has discussed some of these issues in *TUGboat* 10, no. 3. For this article, we only point out that one can concatenate different kinds of data into one control sequence and then `\write` that out. For example, if `\sectionnumber` is a TeX count register containing the current section number, one might augment and alter the above code to

```
\def\sectag{\sec}%
\expandafter\GetMacroMeaning
   \meaning\sectag
   :->\xxsectag\endget
```

```
\def\section #1{%
  ...
  \def\sectionhead{#1}%
  \expandafter\GetMacroMeaning
     \meaning\sectionhead
     :->\xxsectionhead\endget
  \edef\writedata{%
       \xxsectag
       {\number\sectionnumber}%
       {\xxsectionhead}%
       {\noexpand\number\pageno}%
    }%
  \write\outfile\expandafter{\writedata}%
  ...
  }
```

Thus, for section number 3 with title "On $\sin^2 x + \cos^2 x = 1$" and appearing on page 22, the above code will cause

```
    \sec {3}{On $\sin ^2x+\cos ^2x=1$}{22}
```

to be written on `\outfile`. The `\edef` for `\writedata` causes expansion in its replacement text where possible, so the definition of `\writedata` in the case above will be

```
\sec {3}{$\sin ^2x+\cos ^2x=1$}{\number\pageno}
```

Of course, `\sec` here consists of 4 tokens of category 12 (since the whole line passed through `\meaning`), not just one control sequence, and a similar remark holds for the text of the section head. `\number\pageno`, however, has not been sanitized and will be expanded as this token string is written to `\outfile`.

A considerable compaction of all this code can be had by doing the expansions at once (as suggested by Victor Eijkhout). To this end, make the definitions

```
    \def\gobble#1:->{}
    \def\sanitize{%
       \expandafter\gobble\meaning}
    \def\sectag{\sec}
```

and then rework `\section` to

```
    \def\section #1{%
      ...
      \def\sectionhead{#1}%
      \edef\act{%
        \write\outfile{%
          \sanitize\sectag
          {\number\sectionnumber}%
          {\sanitize\sectionhead}%
          {\noexpand\number\pageno}%
          }}%
      \act
      ...
      }
```

Another problem occurs when one wishes to `\write` out long strings of text. Peter Breitenlohner showed in *TUGboat* 11, no. 1 that one could break long strings of text exactly as they had been broken in the source file by activating ⟨*carriage return*⟩s and specifying the `\newlinechar` to be the ⟨*carriage return*⟩. Unfortunately, this method is not transferrable within the current technique exactly because of the sanitizing properties of `\meaning`. `\meaning` will disgorge `^^M` sequences for active ⟨*carriage return*⟩s which cannot in turn be read as ⟨*carriage return*⟩s because the ^ will be of category 'other' instead of 'superscript'.

One way to get over this is to use active `^^M`s as delimiters of line records and `\write` the intervening material line by line to the output file. Here we use a method of Alois Kaelschacht (*TUGboat* 8, no. 2, p. 184; also pointed out by Sonja Maus recently) which allows `\loop`s to contain `\else` clauses.

```
  \def\loop#1\repeat{%
    \def\body{#1\relax\expandafter\body\fi}%
    \body}
  \let\repeat\fi
```

To handle a long piece of text line by line, we define `\ParseLine` to divide the material after it into 2 pieces separated by the first ⟨*carriage return*⟩ in that material.

```
    {\catcode`\^^M=\active
    \gdef\ParseLine#1^^M#2\endParse{%
      \def\FirstLine{#1}%
      \def\Remainder{#2}%
      }}
```

`\Writeit` first turns on the ⟨*carriage return*⟩, then reads the text to be written to a file. Then it runs through a `\loop` until the `\Remainder` text is empty, writing each line with the `\meaning` technique.

```
{\endlinechar=-1
\catcode`\^^M=\active
\catcode`\@=11
\gdef\Writeit{
  \bgroup\catcode`\^^M=\active
  \@Writeit}

\gdef\@Writeit#1\endWriteit{
  \let\FirstLine\empty
  \def\Remainder{#1^^M}
  \loop
    \expandafter
      \ParseLine\Remainder\endParse
    \write\outfile\expandafter{
                 \sanitize\FirstLine}
  \ifx\Remainder\empty\else\repeat
  \egroup}}
```

Of course, this is exactly the kind of nonsense that Breitenlohner avoided with the `\newlinechar` technique, and the 'solution' here still has problems. For one thing, pairs of braces which occur across input lines will cause `\ParseLine` to miss the intervening `^^M`s (since arguments to macros must contain balanced sets of braces). For another, if one also wishes to use the argument to `\Writeit` for something else (such as typesetting here and now), the `^^M`s are now embedded and have not been changed into ⟨*space*⟩ or `\par` as appropriate. It is possible to define the active `^^M` to check ahead for another `^^M` immediately following, changing the pair into `\par` and otherwise inserting ⟨*space*⟩, but at this point we realize we are trying to simulate TeX's mouth behavior with a stomach process and will never be wholly successful. When an input line ends with a control word, TeX's mouth will gobble the end-of-line character, whereas the procedure above will insert an end-of-line ⟨*space*⟩ willy-nilly.

All of which is to say that the `\meaning` technique outlined here should be confined to cases where one is fairly certain that the records to be written are rather short (say, the cases of tables of contents and indexes). In cases where longer records are anticipated, Breitenlohner's technique, accompanied by something analogous to `\protect`, is needed; or one may simply use a verbatim approach if the data is not to be used 'immediately'.

◇ Ron Whitney
TeX Users Group
`rfw@Math.AMS.com`

An Overview of EDMAC: A plain TEX format for critical editions*

John Lavagnino and Dominik Wujastyk

Abstract

EDMAC is a set of plain TEX macros providing the ability to format critical editions of texts in the traditional way, i.e., similar to the Oxford Classical Texts, Teubner, Arden Shakespeare and other series. The principal functions that are added are marginal line numbering and multiple series of footnotes and endnotes keyed to line numbers. While EDMAC's inner workings are necessarily esoteric, it seeks to provide relatively simple macros to enable you to control the exact format of the edition, taking into account the need to vary the format for different sorts of texts.

Contents

1 Introduction

1.1 Overview

The EDMAC macros, together with TEX, provide several important facilities for formatting critical editions of texts in a traditional manner. Major features include:

- automatic stepped line numbering, by page or by chapter;
- sub-lineation within the main series of line numbers;
- variant readings automatically keyed to line numbers;
- multiple series of footnotes and endnotes;
- block or columnar formatting of footnotes.

EDMAC works together with the plain TEX format, and with the exception of footnote-related commands, virtually all plain TEX commands are available for use in the normal way. Other languages and fonts (Sanskrit, Greek, Russian, etc.),[1] can be incorporated.

EDMAC allows the scholar engaged in preparing a critical edition to focus attention wholly on the task of creating the critical text and evaluating the variant readings, text-critical notes and testimonia. TEX and EDMAC will take care of the formatting and visual correlation of all the disparate types of information.

This documentation assumes the "manual" use of EDMAC. But EDMAC is also successfully being used (with TEX, of course) as the formatting engine or "back end" for the output

* This file is Revision: 1.1, Date: 18 Oct 1990 16:09:24.

[1] See *TUGboat* 9, no. 2, pp. 131–151.

of an automatic manuscript collation program. COLLATE runs on the Apple Macintosh, can collate simultaneously up to a hundred manuscripts of any length, and will provide facilities for the scholar to tailor the collation interactively. Version 1.0 of COLLATE is scheduled for distribution from March 1991.[2]

1.2 History

The original version of EDMAC was TEXTED.TEX, written by John Lavagnino in late 1987 and early 1988 for formatting critical editions of English plays.

John passed them on to Dominik Wujastyk who, in September–October 1988, added the footnote paragraphing mechanism, margin swapping and other changes to suit his own purposes,[3] making the style more like that traditionally used for classical texts in Latin and Greek (e.g., the Oxford Classical Texts series). He also wrote some extra documentation and sent the files out to several people. This version of the macros is called EDMAC.TEX.

The present version was developed in the summer of 1990, with the intent of adding necessary features, streamlining and documenting the code, and further generalizing it to make it easily adaptable to the needs of editors in different disciplines. John did most of the general reworking and documentation, with the financial assistance of the Division of the Humanities and Social Sciences, California Institute of Technology. Dominik adapted the code to the conventions of Frank Mittelbach's doc option, and added some documentation, multiple-column footnotes, cross-references, and crop marks.

2 How to use EDMAC

2.1 Introduction

All you need to do to invoke EDMAC is to include the line `\input EDMAC` at the top of your document, and to have the file EDMAC.TEX somewhere on your disk that is "visible" to TEX for input. If you are going to use it frequently, as will certainly be the case if you are doing a real edition, you will find it convenient to compile it into a TEX format file, loading it after PLAIN.TEX and any other private macros.

EDMAC is a three-pass system. Although your textual apparatus and line numbers will be printed even on the first run, it takes two more passes through TEX to be sure that everything gets to its right place. Any changes you make to the input file may similarly require three passes to get everything to the right place, if the changes alter the number of lines or notes. EDMAC will tell you that you need to make more runs, when it notices, but it does not expend the labor to check this thoroughly. If you have problems with a line or two misnumbered at the top of a page, try running TEX once or twice more.

A file may mix *numbered* and *unnumbered* text. Numbered text is printed with marginal line numbers and can include footnotes and endnotes that are referenced to those line numbers: this is how you'll want to print the text that you're editing. Unnumbered text is not printed with line numbers, and you can't use EDMAC's note commands with it: this is appropriate for introductions and other material added by the editor around the edited text.

2.2 General markup

`\beginnumbering` Each section of numbered text must be preceded by `\beginnumbering` and followed

[2] Contact COLLATE's author, Peter Robinson, at The Computers and Manuscripts Project, Oxford University Computing Service, 13 Banbury Road, Oxford OX2 6NN, England. Janet: `peterr@uk.ac.oxford.vax`.

[3] These macros were used to format the Sanskrit text in *Metarules of Pāṇinian Grammar* by Dominik Wujastyk (Groningen, in press).

`\endnumbering` by `\endnumbering`:

```
\beginnumbering
⟨text⟩
\endnumbering
```

The `\beginnumbering` macro resets the line number to zero, reads an auxiliary file called ⟨*filename*⟩`.nn` (where `nn` is 1 for the first section, 2 for the second section, and so on), and then creates a new version of this auxiliary file to collect information during this run. The first instance of `\beginnumbering` also opens a file called ⟨*filename*⟩`.end` to receive the text of the endnotes. `\endnumbering` closes the ⟨*filename*⟩`.nn` file.

If the line numbering of a text is to be continuous from start to end, then the whole text will be typed between one pair of `\beginnumbering` and `\endnumbering` commands. But your text will most often contain chapter or other divisions marking sections that should be independently numbered, and these will be appropriate places to begin new numbered sections. EDMAC has to read and store in memory a certain amount of information about the entire section when it encounters a `\beginnumbering` command, so it speeds up the processing and reduces memory use when a text is divided into a larger number of sections (at the expense of multiplying the number of external files that are generated).

`\pstart` `\pend` Within a numbered section, each paragraph of numbered text must be marked using the `\pstart` and `\pend` commands:

```
\pstart
⟨paragraph of text⟩
\pend
```

Text that appears within a numbered section but isn't marked with `\pstart` and `\pend` will not be numbered.

The following example shows the proper section and paragraph markup:

```
\beginnumbering
\pstart
This is a sample paragraph, with
lines numbered automatically.
\pend

\pstart
This paragraph too has its
lines automatically numbered.
\pend

The lines of this paragraph are
not numbered.

\pstart
And here the numbering begins
again.
\pend
\endnumbering
```

```
1  This is a sample paragraph
2  with lines numbered
3  automatically.
4  This paragraph too
5  has its lines automatically
6  numbered.
   The lines of this paragraph
   are not numbered.
7  And here the numbering
8  begins again.
```

`\autopar` You can use `\autopar` to avoid the nuisance of this paragraph markup and still have every paragraph automatically numbered, in this manner:

```
\begingroup
  \beginnumbering
  \autopar

  A paragraph of numbered text.

  Another paragraph of numbered text.

  \endnumbering
\endgroup
```

```
1  A paragraph of numbered
2  text.
3  Another paragraph of
4  numbered text.
```

`\autopar` fails, however, on paragraphs that start with a `{` or with any other command that starts a new group before it generates any text. Such paragraphs still need to be started explicitly using `\pstart`.

2.3 The apparatus

`\text`

Within numbered paragraphs, all footnotes and endnotes are generated by forms of the `\text` macro:

`\text{`⟨*lemma*⟩`}{`⟨*commands*⟩`}/`

The ⟨*lemma*⟩ argument is the lemma in the main text: `\text` both prints this as part of the text, and makes it available to the ⟨*commands*⟩ you specify to generate notes. The `/` at the end terminates the command.

For example:

```
I saw my friend \text{Smith}
{\afootnote{Jones C, D.}}/
on Tuesday.
```

```
1 I saw my friend
2 Smith on Tuesday.
________
2 Smith] Jones C, D.
```

The lemma `Smith` is printed as part of this sentence in the text, and is also made available to the footnote that specifies a variant, `Jones C, D`. The footnote macro is supplied with the line number at which the lemma appears in the main text.

The ⟨*lemma*⟩ may contain further `\text` commands. This makes it possible to print an explanatory note on a long passage together with notes on variants for individual words within the passage. For example:

```
\text{I saw my friend
  \text{Smith}{\afootnote{Jones
  C, D.}}/ on Tuesday.}
  {\bfootnote{The date was
  July 16, 1954.}
}/
```

```
1 I saw my friend
2 Smith on Tuesday.
________
2 Smith] Jones C, D.
________
1–2 I saw my friend
Smith on Tuesday.] The
date was July 16, 1954.
```

However, `\text` cannot handle overlapping but unnested notes—for example, one note covering lines 10–15, and another covering 12–18; a `\text` that starts in the ⟨*lemma*⟩ argument of another `\text` must end there, too. (The `\lemma` and `\linenum` commands may be used to generate overlapping notes if necessary.)

Commands used in `\text`'s second argument The second argument of the `\text` macro, ⟨*commands*⟩, may contain a series of subsidiary commands that generate various kinds of notes. The braces around ⟨*commands*⟩ are optional, unless this instance of `\text` appears within the argument of another instance of `\text`.

`\afootnote`
`\bfootnote`
`\cfootnote`
`\dfootnote`
`\efootnote`

Five separate series of footnotes are maintained; when all five are used, the `a` notes appear in a layer just below the main text, followed by the rest in turn, down to the `e` notes at the bottom. These are the main macros that you will use to construct the critical apparatus of your text. `EDMAC` provides five layers of notes in the belief that this will be adequate for the most demanding editions. But it is not hard to add further layers of notes to EDMAC should they be required.

`\aendnote` `\bendnote` `\cendnote` `\dendnote` `\eendnote`

EDMAC also maintains five separate series of endnotes. By default none of them are printed: you must use the `\doendnotes` macro described below (p. 629) to call for this at the appropriate point in your document.

`\lemma`

Sometimes you want to change the lemma that gets passed to the notes. You can do this by using `\lemma` within the second argument to `\text`, before the note commands.

`\lemma{`⟨*alternative lemma*⟩`}`

The most common use of this command is to abbreviate the lemma that's printed in the notes. For example:

```
\text{I saw my friend
  \text{Smith}{\afootnote{Jones
  C, D.}}/ on Tuesday.}
  {\lemma{I \dots\ Tuesday.}
  \bfootnote{The date was
  July 16, 1954.}
}/
```

1 I saw my friend
2 Smith on Tuesday.

2 Smith] Jones C, D.

1–2 I . . . Tuesday.]
The date was July 16, 1954.

`\linenum`

You can use `\linenum` to change the line numbers passed to the notes. The notes are actually given seven numbers: the page, line, and sub-line number for the start of the lemma; the same three numbers for the end of the lemma; and the font family number for the lemma. As argument to `\linenum`, you specify those seven numbers in that order, separated by vertical bars (the | character). However, you can retain the value computed by EDMAC for any number by simply omitting it; and you can omit a sequence of vertical bars at the end of the argument. For example, `\linenum{|||23}` changes one number, the ending page number of the current lemma.

This command doesn't change the marginal line numbers in any way; it changes the numbers passed to the footnotes. Its use comes in situations that `\text` has trouble dealing with for whatever reason. If you need notes for overlapping passages that aren't nested, for instance, you can use `\lemma` and `\linenum` to generate such notes despite the limitations of `\text`. If the ⟨*lemma*⟩ argument to `\text` is extremely long, you may run out of memory; here again you can specify a note with an abbreviated lemma using `\lemma` and `\linenum`. The numbers used in `\linenum` need not be entered manually; you can use the "`x-`" cross-referencing commands below (p. 629) to compute them automatically.

Changing the names of these commands The default commands for generating the apparatus have been given rather bland names, because editors in different fields have widely divergent notions of what sort of notes are required, where they should be printed, and what they should be called. But this doesn't mean you should type `\afootnote` when you'd rather say something you find more meaningful, like `\variant`. We recommend that you create a series of such aliases and use them instead of the names chosen here; all you have to do is put commands of this form at the start of your file:

```
\let\variant=\afootnote
\let\explanatory=\bfootnote
\let\trivial=\aendnote
```

It is also possible to define aliases for `\text`, which can be easier to type. You can make a single character substitute for `\text` by saying this:

```
\catcode`\<=\active
\let<=\text
```

Then you can say `<{Smith}\afootnote{Jones}/`. This of course destroys the ability to use `<` in any new macro definitions, so it requires some care.

Changing the character at the end of the command requires more work:

```
\catcode'\<=\active
\def\xtext#1#2>{\text{#1}{#2}/}
\let<=\xtext
```

This allows you to say `<{Smith}\afootnote{Jones}>`.

These aliases can't be nested: if you want to use a `\text` within the first argument of another `\text`, the outer `\text` can use an alias but not the inner. For example,

```
<{a \text{big}{\afootnote{bin}}/ difference}\bfootnote{no difference}>
```

2.4 Lineation commands

`\lineation` EDMAC can number lines either by page or by section; you specify this using the `\lineation{`⟨*arg*⟩`}` macro, where ⟨*arg*⟩ is either `page` or `section`. You may only use this command at places when numbering is not in effect; you can't change the lineation system within a section. You can change it between sections: they don't all have to use the same lineation system. The line-of-section system is used by default.

`\linenummargin` The marginal line numbers will be printed in the `left`, `right`, `inner`, or `outer` margin, depending on which you specify as argument to this command. By default, line numbers appear in the left margin. You can change this whenever you're not in the middle of making a paragraph. E.g.: `\linenummargin{inner}`.

`\firstlinenum` `\linenumincrement` `\firstsublinenum` `\sublinenumincrement` You set these counters to control which lines are printed with marginal numbers. `\firstlinenum` is the first line in a section to number, and `\linenumincrement` is the increment between numbered lines. The other parameters do the same for sub-lines. By default all these counters are set equal to 5.

`\leftlinenum` `\rightlinenum` `\linenumsep` `\numlabfont` These parameters control the appearance of marginal line numbers. You can redefine `\leftlinenum` and `\rightlinenum` to change the way marginal line numbers are printed; the default values print the number in font `\numlabfont` at a distance `\linenumsep` from the text.

2.5 Changing the line numbers

`\startsub` `\endsub` These macros turn sub-lineation on and off. When sub-lineation is in effect, the line number counter is frozen and the sub-line counter advances instead. If one of these commands appears in the middle of a line, it doesn't take effect until the next line; in other words, a line is counted as a line or sub-line depending on what it started out as, even if that changes in the middle.

`\setline` `\advanceline` These commands may be used to change the line number (or the sub-line number, if sub-lineation is currently on). They change both the marginal line numbers and the line numbers passed to the notes. `\setline` takes one argument, the value to which you want the line number set; it must be 0 or greater. `\advanceline` takes one argument, an amount that should be added to the current line number; it may be positive or negative.

2.6 Alternate footnote formatting

`\notenumfont` `\lemmafont` `\notetextfont` These commands select the fonts used in printing all the layers of notes. The `\notenumfont` and `\notetextfont` macros take no arguments; they should be equated to the appropriate fonts using `\let`. `\lemmafont` is a macro that takes one argument—the cluster of line numbers passed to the note commands, a cluster that ends with a number indicating what font family was in effect at the start of the lemma; `\lemmafont` selects the appropriate font for the note using that family number. See the full documentation for more details. What EDMAC does is to use these macros in a default footnote format macro called `\normalfootfmt`. The footnote formats for each of the layers `a` to `e` are `\let` equal to `\normalfootfmt`.

But it is also likely that you might want to have different fonts for just, say, the note numbers in layers `a` and `b` of your apparatus. To do this, make two copies of the `\normalfootfmt` macro (see the `arden.sty` example, p. 636 below) or `\twocolfootfmt`, or other appropriate macro ending in `-footfmt`, depending on what footnote format you have selected, and give these macros the names `\afootfmt` and `\bfootfmt`. Then, in these new macros, change the font specifications (and spacing, or whatever) to your liking.

`\footparagraph` `\foottwocol` `\footthreecol` All footnotes will normally be formatted as a series of separate paragraphs in one column. But there are three other formats available for notes, and using these macros you can select a different format for a series of notes. `\footparagraph` formats all the footnotes of a series as a single paragraph; `\foottwocol` formats them as separate paragraphs, but in two columns; `\footthreecol`, in three columns. Each of these macros takes one argument: a letter (between `a` and `e`) for the series of notes you want changed.

You should set `\hsize` for the text, and the `\baselineskip` of the footnotes (`\afootbaselineskip`, etc.), before you call any of these macros, because their action depends on those values; too much or too little space will be allotted for the notes on the page if these macros use the wrong values.

2.7 Crop marks

Publishers usually like crop marks on the camera-ready copy for works of this kind, so a facility for generating them has been incorporated into EDMAC.

Publishers specify crop marks (or trim lines, etc.) in terms of two dimensions, height and width, and they also usually specify back and head margins.

The "head margin" is the distance between the top of the printed text and the top crop marks; it is normally measured from the top of the running head, `plain` TEX's `\headline`. The "back margins" (or "gutter margins") are the right margins of even-numbered pages, and the left margins of odd-numbered pages. If you hold a book open in front of you, they are the margins in the middle of the opening.

`\cropsetup` If you want to have crop marks, and to control the back and head margins, you issue the `\cropsetup` macro. It takes four parameters (see Figure 1):

1. the vertical distance between crop marks,
2. the horizontal distance between crop marks,
3. the head margin, and
4. the back margin.

In order to calculate these dimensions properly, EDMAC has to use the `\hsize` of the page, as well as information about the height at which the `\headline` floats above the main text (which is set in `plain` TEX's `\makeheadline` macro). EDMAC performs these calculations when you issue the `\cropsetup` command. Therefore, it is important that you set the `\hsize` and make any changes to `\makeheadline` *before* you issue the `\cropsetup` command. If you do change these values, issue the `\cropsetup` command again.

`\headlinefont` `\magicvskip` In particular, if the `\headline` is going to be set at a different height from the top of the text, or in a different font, you can change the appropriate values easily by using `\headlinefont` and `\magicvskip`. The former is what you would expect; just `\let` it to be whatever font you like (the font macro should include a definition of an appropriate `\strutbox` for that font). The `\magicvskip` gives you direct access

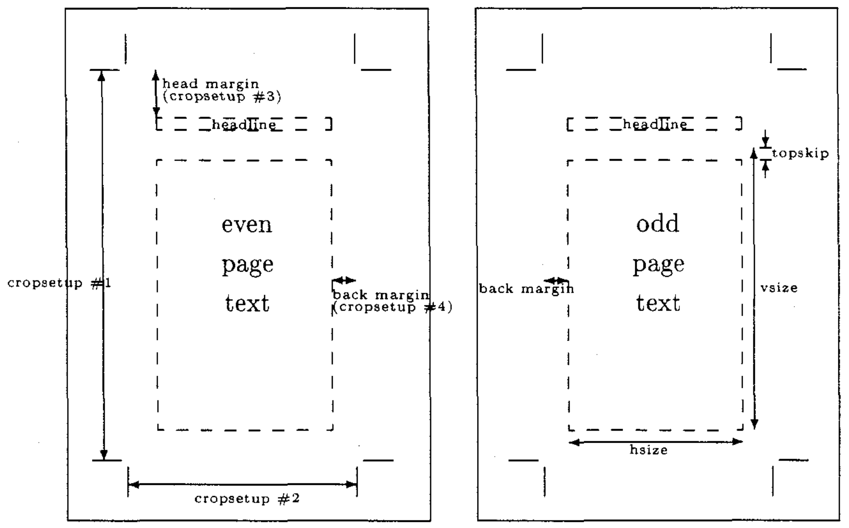


Figure 1: Crop marks, back and head margins.

to what Knuth calls a "magic constant" on p. 255 of *The TEXbook*. For `plain` TEX, `\magicvskip` is −22.5 pt, but you can change this if you want the `\headline` higher or lower than the default. See the full documentation for more details.

Apart from this stricture that the `\cropsetup` command should follow any changes in `\hsize` and the `\headline`, there is no relation (other than visual) between the crop marks and the `\hsize` and `\vsize`. You can vary any of these dimensions independently, without affecting any other. Your publisher will almost always want the `\hsize` and `\vsize` to be a few picas smaller than the horizontal and vertical distances between crop marks. And if you want to shift the whole of your printed page about on the paper, use `\hoffset` and `\voffset` as described in *The TEXbook*, 251, or use the facilities of your DVI translator.

`\cropwidth` `\cropgap` `\cropwidth` and `\cropgap` define the thickness of the rules used for drawing crop marks and the gap by which crop marks don't cross; as before, if you change either, do so before using the `\cropsetup` macro.

If, for example, you want your text to have a back margin, for two-sided printing, but you don't want crop marks, just set `\cropwidth=0pt`.

2.8 Endnotes

`\doendnotes` `\endprint` `\doendnotes` closes the `.end` file, if it's open, and prints one series of endnotes, as specifed by a series-letter argument, e.g., `\doendnotes{a}`. `\endprint` is the macro that's called to print each note. It uses `\notenumfont`, `\lemmafont`, and `\notetextfont` to select fonts, just as the footnote macros do (see p. 627 above).

`\noendnotes` If you aren't going to have any endnotes, you can say `\noendnotes` in your file, before the first `\beginnumbering`, to suppress the generation of an unneeded `.end` file.

2.9 Cross referencing

EDMAC provides a simple cross-referencing facility that allows you to mark places in the text with labels, and generate page and line number references to those places elsewhere in the text using those labels.

`\label` First you place a label in the text using the command `\label{foo}`. "`foo`" can be anything you like, including letters, numbers, punctuation, or a combination—anything but spaces; you might say `\label{toves-3}`, for example.

\pageref \lineref \sublineref Elsewhere in the text, either before or after the \label, you can refer to its location by saying \pageref{foo}, or \lineref{foo}, or \sublineref{foo}. These commands will produce, respectively, the page, line and sub-line on which the \label{foo} command occurred.

A \label command may appear in the main text, or in the first argument of \text. But \pageref, \lineref and \sublineref commands can be used in the apparatus to refer to \labels in the text.

The \label command works by writing macros to an .aux file (which will only be created if you are actually using some of these commands). Clearly, then, you will need to process your document through TeX twice in order for the references to be resolved.

You will be warned if you say \label{foo} and foo has been used as a label before. The ref commands will return references to the last place in the file marked with this label. You will also be warned if a reference is made to an undefined label. (This will also happen the first time you process a document after adding a new \label command: the auxiliary file will not have been updated yet.)

\xpageref \xlineref \xsublineref However, there are situations in which you'll want EDMAC to return a number without displaying such a warning: if you want to use the reference in a context where TeX is looking for a number, such a warning will lead to a complaint that the number is missing. This is the case for references used within the argument to \linenum, for example. For this situation, these variants of the reference commands, with the x prefix, are supplied: the only operations they perform are ones that TeX can do in its "mouth." They have these limitations: they will not tell you if the label is undefined, and they must be preceded in the file by at least one of the four other cross-reference commands—e.g., a \label{foo} command, even if you never refer to that label—*since those commands can all do the necessary processing of the* .aux file, and these cannot.

2.10 Miscellaneous

\extensionchars When EDMAC assembles the name of the auxiliary file for a section, it prefixes \extensionchars to the section number. By default this is empty, but you can add some characters to help distinguish these files if you like; what you use is likely to be system-dependent. If, for example, you said \def\extensionchars{!}, then you would get temporary files called jobname.!1, jobname.!2, etc.

2.11 Known bugs

The plain TeX \footnote command will work only within unnumbered text; within numbered text it will wreak havoc.

One seemingly small point that does make a difference to EDMAC is that the definition of each font should include a definition of the appropriate \strut and \strutbox. Like plain TeX itself, EDMAC uses the height of a \strutbox in one or two places (the crop marks, the alignment of the top line of footnotes, etc.), and if you change the size of your fonts, but don't change the size of the \strutbox too, then there will be discrepancies in the spacing. For an example of how to do this, see the definition of \eightpoint on p. 415 of *The TeXbook.*

\parshape cannot be used within numbered text, except in a very restricted way (see the full documentation for more details).

Critical editions, like dictionaries, present a great deal of categorized information in a densely compressed form. Success in making the results legible typically depends on the carefully planned use of a large variety of fonts. There are several places where EDMAC suffers from the lack of a general font selection scheme such as Mittelbach and

Schöpf's.[4] We look forward to a time when it will be possible to rationalize EDMAC's font calls, and bring them into line with such a general scheme.

Any help, suggestions and corrections gratefully received.

3 Examples

In the following examples, the command `\input edmac` has been included for completeness although, as mentioned before, it is usually more convenient to include the EDMAC macros in a TeX format, to be invoked with a command such as

`tex &edmac` ⟨*filename*⟩.

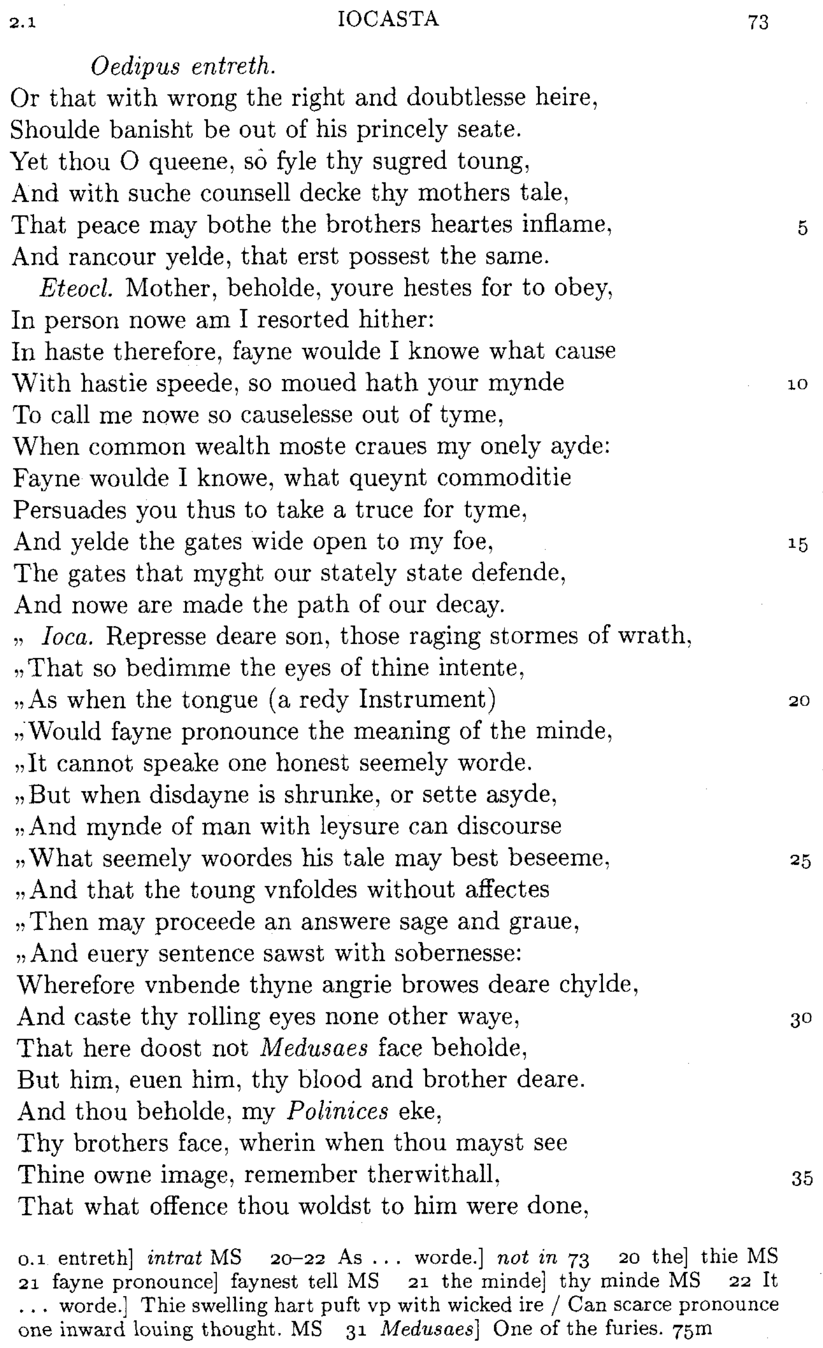
3.1 Gascoigne

The first example is taken from an edition of George Gascoigne's *A Hundreth Sundrie Flowres* that is being prepared by G. W. Pigman III. Figure 2 shows the result of setting the text with EDMAC.

The main input file first calls for a file of initial definitions, called `gg.tex`. This file, shown below, demonstrates how EDMAC macros may be customized to give detailed control over the final format.

```
% parameters for edition of Gascoigne's {\it A Hundreth Sundrie Flowres}.
\ifx\ggloaded\relax\endinput\else\let\ggloaded=\relax\fi
\noendnotes
\makeatletter
\def\eightpoint{\def\rm{\fam0\eightrm}%
  \def\it{\fam\itfam\eightit}%
  \setbox\strutbox=\hbox{\vrule height7pt depth2pt width\z@}%
  \rm}
\font\poemnumfont=cmmi12 \font\titlefont=cmr12 \font\ninerm=cmr9
\font\nineit=cmti9       \font\eightrm=cmr8    \let\headfont=\eightrm
\font\eightit=cmti8      \let\headit=\eightit  \font\sixrm=cmr6
\font\foliofont=cmmi8    \let\os=\foliofont    \let\numlabfont=\foliofont
%
\firstsublinenum=1000
\hoffset=1.25in \voffset=1.25in \hsize=24pc \vsize=488pt
%
\frenchspacing \parskip=0pt \hyphenpenalty=1000
%
\def\makeheadline{\vbox to 0pt{\vskip-16.5pt
  \line{\vbox to8.5pt{}\the\headline}\vss}\nointerlineskip}
\nopagenumbers
%
\newif\ifnolinenums     % true if you want no line number in the notes
\def\nolinenums{\global\nolinenumstrue}
\def\linenums{\global\nolinenumsfalse}
\newif\ifpoemnum        % true if you want to print the poem number in
                        % the notes
\def\nopoemnum{\global\poemnumfalse}
\newif\ifdbpoemnum      % ditto for poems with two numbers, e.g., 64 (v)
\def\nodbpoemnum{\global\dbpoemnumfalse}
\newif\ifactnum         % ditto for act/scene numbers
\def\noactnum{\global\actnumfalse}
%
\newcount\poemnumber
\def\poem#1{\poemnumber=#1\poemnumtrue\parindent=0pt
  \centerline{{\poemnumfont#1}}\vskip12pt}
```

[4] See *TUGboat* 10, no. 2, pp. 222–238; 11, no. 1, pp. 91–97.


2.1 IOCASTA 73

Oedipus entreth.

Or that with wrong the right and doubtlesse heire,
Shoulde banisht be out of his princely seate.
Yet thou O queene, so fyle thy sugred toung,
And with suche counsell decke thy mothers tale,
That peace may bothe the brothers heartes inflame,
And rancour yelde, that erst possest the same.
Eteocl. Mother, beholde, youre hestes for to obey,
In person nowe am I resorted hither:
In haste therefore, fayne woulde I knowe what cause
With hastie speede, so moued hath your mynde
To call me nowe so causelesse out of tyme,
When common wealth moste craues my onely ayde:
Fayne woulde I knowe, what queynt commoditie
Persuades you thus to take a truce for tyme,
And yelde the gates wide open to my foe,
The gates that myght our stately state defende,
And nowe are made the path of our decay.
„ *Ioca.* Represse deare son, those raging stormes of wrath,
„That so bedimme the eyes of thine intente,
„As when the tongue (a redy Instrument)
„Would fayne pronounce the meaning of the minde,
„It cannot speake one honest seemely worde.
„But when disdayne is shrunke, or sette asyde,
„And mynde of man with leysure can discourse
„What seemely woordes his tale may best beseeme,
„And that the toung vnfoldes without affectes
„Then may proceede an answere sage and graue,
„And euery sentence sawst with sobernesse:
Wherefore vnbende thyne angrie browes deare chylde,
And caste thy rolling eyes none other waye,
That here doost not *Medusaes* face beholde,
But him, euen him, thy blood and brother deare.
And thou beholde, my *Polinices* eke,
Thy brothers face, wherin when thou mayst see
Thine owne image, remember therwithall,
That what offence thou woldst to him were done,

0.1 entreth] *intrat* MS 20–22 As . . . worde.] *not in* 73 20 the] thie MS 21 fayne pronounce] faynest tell MS 21 the minde] thy minde MS 22 It . . . worde.] Thie swelling hart puft vp with wicked ire / Can scarce pronounce one inward louing thought. MS 31 *Medusaes*] One of the furies. 75m

Figure 2: Output from `iocasta.tex`.

```
%
\newcount\dbpoemnumone  \newcount\dbpoemnumtwo
\def\dbpoem#1#2{\dbpoemnumtrue\dbpoemnumone=#1\dbpoemnumtwo=#2
  \parindent=0pt\par
  \centerline{{\poemnumfont#1} {\titlefont
   (\romannumeral#2)}}\nointerlineskip \vskip12pt}
%
\newcount\actnumber \newcount\scenenumber
\def\act #1 #2 #3{\actnumtrue\actnumber=#1\scenenumber=#2
  \parindent=0pt\vskip24pt plus12pt minus3pt\hrule height0pt\relax
  \pstart\startsub\centerline{\rm#3}\pend\endsub
  \mark{{\os#1:#2}}\nobreak \vskip 12pt plus3pt minus3pt}
% : is . in \os
%
\def\rightlinenum{\if@bypage\ifnum\line@num<10\kern.5em\fi\else
\ifnum\line@num<10\kern1em\else\ifnum\line@num<100
  \kern.5em\fi\fi\fi\kern.5em\numlabfont\the\line@num
  \ifnum\subline@num>0:\the\subline@num\fi}
\def\leftlinenum{\numlabfont\the\line@num
  \ifnum\subline@num>0:\the\subline@num\fi \kern.5em}
\linenummargin{outer}
\lineation{page}
\def\ggfootfmt#1#2#3{%
  \eightpoint
  \let\par=\endgraf
  \rightskip=0pt \leftskip=0pt
  \parindent=0pt \parfillskip=0pt plus 1fil
  \printlines#1\ifnolinenums\relax\else\enskip\fi {\rm
    #2\def\@tempa{#2}\ifx\@tempa\empty
    \else]\enskip\fi#3\penalty-10\hskip 1em plus1em minus.4em\relax}}
% : is . in \os
%
\def\printlines#1|#2|#3|#4|#5|#6|#7{%
\eightpoint
%
%  Do nothing if no line numbers are wanted.
  \ifnolinenums
%
%  We are printing line numbers. Go into the old-style-digits font.
  \else
    \begingroup
       \os
%
%  First, print poem number or act/scene number.
    \ifpoemnum          % a poem
      \the\poemnumber:%
      \global\poemnumfalse
    \fi
    \ifdbpoemnum                % a poem with two numbers, e.g. 64 (xii)
      \the\dbpoemnumone\space
          {\rm(\romannumeral\the\dbpoemnumtwo).}%
      \global\dbpoemnumfalse
    \fi
    \ifactnum                   % a play (act/scene)
      \the\actnumber:\the\scenenumber:%
      \global\actnumfalse
    \fi
%  Now do the line numbers.  To simplify the logic here we use a lot of
%  counters to tell us which numbers need to get printed (using 1 for
% yes, 0 for no).  The assignments are: 0 for page numbers; 2 for
% starting subline; 4 for ending line; 6 for ending subline; and 8 for
% dash between the starting and ending groups.  There's no counter for
% the line number because it's always printed.
% We print page numbers only if:
%      --- we're doing by-page lineation, and
%      --- the ending page number is different from
%          the starting page number.
   \count0=0 \count8=0
   \if@bypage
      \ifnum#4=#1 \else
         \count0=1
         \count8=1
      \fi
   \fi
%
%  The ending line number is printed if:
%        --- we're printing the ending page number, or
%        --- it's different from the starting line number.
   \count4=\count0
   \ifnum#2=#5 \else
       \count4=1
       \count8=1
   \fi
%
%  The starting subline is printed if it's nonzero.
   \count2=0
   \ifnum#3=0 \else
       \count2=1
   \fi
%
%  The ending subline number is printed if it's nonzero and:
%      --- different from the starting subline number, or
%      --- the endline is being printed.
   \count6=0
   \ifnum#6=0 \else
       \ifnum#6=#3
          \count6=\count4
       \else
          \count6=1
          \count8=1
       \fi
   \fi
%
%  Now we're ready to print it all, based on our counter values.  The
%  only subtlety left here is when to print a : between numbers.
%  But the only instance in which this is tricky is for ending subline
%  number: it could be coming after the starting subline number (in
%  which case we want only the dash) or after an ending line number
%  (in which case we need to insert a :).
   \ifodd\count0 #1:\fi
   #2%
   \ifodd\count2 :#3\fi
   \ifodd\count8 {\rm --}\fi
   \ifodd\count0 #4:\fi
   \ifodd\count4 #5\fi
   \ifodd\count6 \ifodd\count4:\fi #6\fi
%
   \endgroup  % end of \os font
```

```
  \fi\ignorespaces}
%
% Now reset the \afootnote parameters and macros:
\afootbaselineskip=9pt
\footparagraph{a}
\let\afootfmt=\ggfootfmt
\dimen\afootins=\vsize
\skip\afootins=3pt plus9pt
\def\ggfootstart#1{\vskip\skip\afootins}
\let\afootstart=\ggfootstart
\def\title{\pstart\startsub\let\par=\endtitle}
\def\endtitle{\pend\endsub}
\def\verseskip{\vskip6pt plus6pt}
\def\speaker#1{\pstart\parindent=1em\let\par=\pend
  {\tenit{#1}}\hbox to 1ex{}\ignorespaces}
\def\sen{\leavevmode\lower1ex\hbox{\tenrm''}}
\def\senspeak#1{\pstart\obeylines\setbox0=\hbox{\tenrm''}\leavevmode
  \lower1ex\copy0\kern-\wd0\hskip1em{\tenit{#1}}\hbox to1ex{}\ignorespaces}
\def\speak#1{\pstart\obeylines\hskip1em{\tenit{#1}}\hbox to
  1ex{}\ignorespaces}
\def\nospeaker{\parindent=0em\pstart\let\par=\pend}
\def\nospeak{\pstart\obeylines}
\def\stage#1{\pstart\startsub\parindent=0pt\hangindent=3em\hangafter=0
  {\tenit#1}\let\par=\endstage}
\let\endstage=\endtitle
\def\motto#1{\pstart\startsub\centerline{{\tenit#1}}\pend\endsub}
\def\finis#1{\pstart\startsub\smallskip\centerline{{\tenit#1}}
  \let\par=\endfinis}
\let\endfinis=\endtitle
\def\initials#1{\pstart\line{\hfil{\it #1}\quad}\let\par=\pend}
\makeatother
```

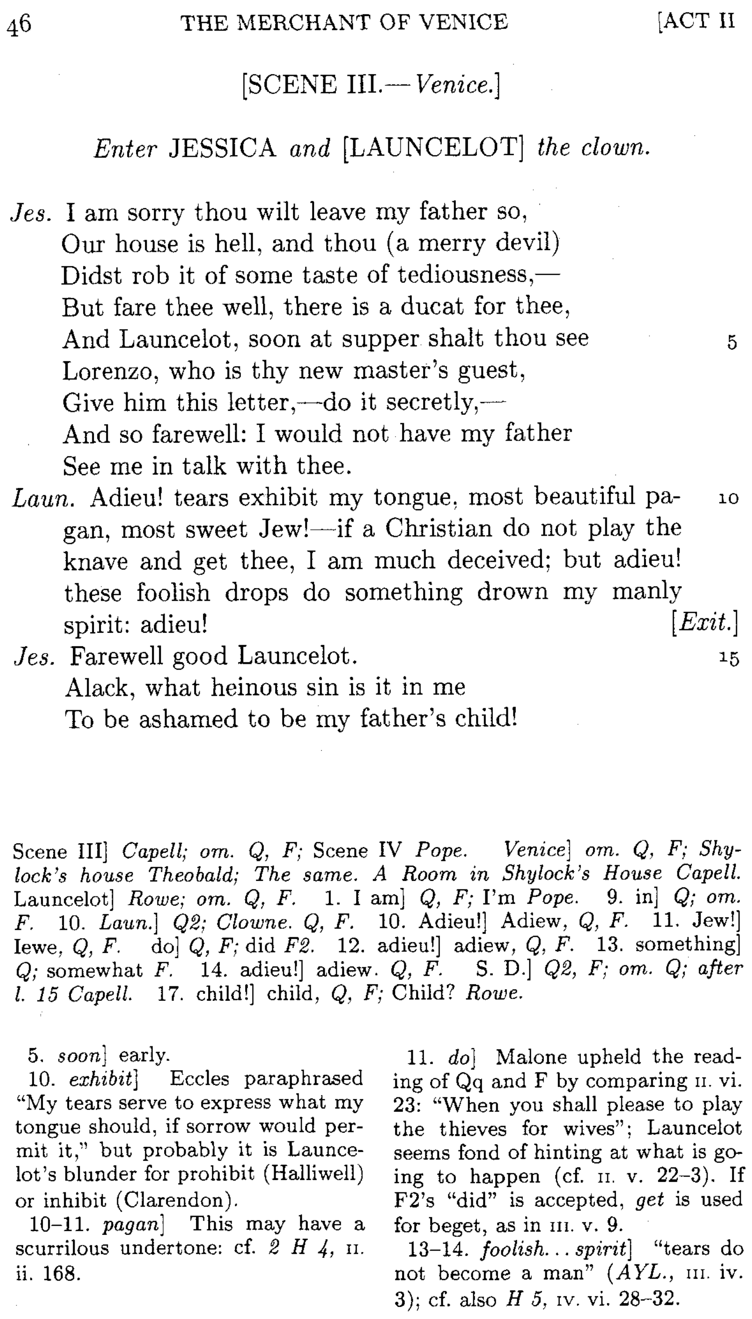
With these definitions, the actual input file, `iocasta.tex`, is relatively simple:

```
\input edmac
\input gg
\parindent=0pt
\pageno=73
\mark{{\os2:1}}
\headline={\ifnum\pageno>61\ifodd\pageno
    \rlap{\foliofont\botmark}\hfil\headfont
    IOCASTA\hfil\llap{\foliofont\folio}%
  \else
    \rlap{\foliofont\folio}\hfil\headfont
    IOCASTA\hfil\llap{\foliofont\botmark}\fi
  \else\hfil\fi}

\beginnumbering

\stage{Oedipus \text{entreth}\afootnote{{\it intrat} MS}/.}

\nospeak
Or that with wrong the right and doubtlesse heire,
Shoulde banisht be out of his princely seate.
Yet thou O queene, so fyle thy sugred toung,
And with suche counsell decke thy mothers tale,
That peace may bothe the brothers heartes inflame,
And rancour yelde, that erst possest the same.
\pend

\speak{Eteocl.} Mother, beholde, youre hestes for to obey,
In person nowe am I resorted hither:
In haste therefore, fayne woulde I knowe what cause
With hastie speede, so moued hath your mynde
To call me nowe so causelesse out of tyme,
When common wealth moste craues my onely ayde:
Fayne woulde I knowe, what queynt commoditie
Persuades you thus to take a truce for tyme,
And yelde the gates wide open to my foe,
The gates that myght our stately state defende,
And nowe are made the path of our decay.
\pend

\senspeak{Ioca.}Represse deare son, those raging stormes of wrath,
\sen That so bedimme the eyes of thine intente,
\text{\sen As when \text{the}\afootnote{thie MS}/ tongue (a redy Instrument)
\sen Would \text{fayne pronounce}\afootnote{faynest tell MS}/ the meaning %
of \text{the minde}\afootnote{thy minde MS}/,
\sen \text{It}\lemma{It \dots\ worde.}\afootnote{Thie swelling hart %
puft vp with wicked ire / Can scarce pronounce one inward louing %
thought. MS}/ cannot speake one honest seemely worde.}\lemma{%
As \dots\ worde.}\afootnote{{\it not in\/} \os73}/
\sen But when disdayne is shrunke, or sette asyde,
\sen And mynde of man with leysure can discourse
\sen What seemely woordes his tale may best beseeme,
\sen And that the toung vnfoldes without affectes
\sen Then may proceede an answere sage and graue,
\sen And euery sentence sawst with sobernesse:
Wherefore vnbende thyne angrie browes deare chylde,
And caste thy rolling eyes none other waye,
That here doost not \text{{\it Medusaes\/}}%
\afootnote{One of the furies. {\os75}m}/ face beholde,
But him, euen him, thy blood and brother deare.
And thou beholde, my {\it Polinices\/} eke,
Thy brothers face, wherin when thou mayst see
Thine owne image, remember therwithall,
That what offence thou woldst to him were done,
\pend
\endnumbering
\bye
```

3.2 Shakespeare

The following text illustrates another input file of moderate complexity, with two layers of annotation in use. The example is taken from the Arden *Merchant of Venice*. First, the file `arden.sty` contains a set of font definitions and format specifications:

```
\makeatletter
% Load small fonts: (cf. TeXbook, p.413-415):
\font\eightrm=cmr8 \font\eighti=cmmi8  \skewchar\eighti='177
\font\eightsy=cmsy8 \skewchar\eightsy='60 \font\eightbf=cmbx8
\font\eighttt=cmtt8 \hyphenchar\eighttt=-1 % inhibit hyphenation
\font\eightsl=cmsl8 \font\eightit=cmti8
\font\sixrm=cmr8  \font\sixi=cmmi8    \skewchar\sixi='177
```



46 THE MERCHANT OF VENICE [ACT II

[SCENE III.— *Venice.*]

Enter JESSICA *and* [LAUNCELOT] *the clown.*

Jes. I am sorry thou wilt leave my father so,
Our house is hell, and thou (a merry devil)
Didst rob it of some taste of tediousness,—
But fare thee well, there is a ducat for thee,
And Launcelot, soon at supper shalt thou see
Lorenzo, who is thy new master's guest,
Give him this letter,—do it secretly,—
And so farewell: I would not have my father
See me in talk with thee.

Laun. Adieu! tears exhibit my tongue, most beautiful pagan, most sweet Jew!—if a Christian do not play the knave and get thee, I am much deceived; but adieu! these foolish drops do something drown my manly spirit: adieu! [*Exit.*]

Jes. Farewell good Launcelot.
Alack, what heinous sin is it in me
To be ashamed to be my father's child!

Scene III] *Capell; om. Q, F;* Scene IV *Pope.* Venice] *om. Q, F; Shylock's house Theobald; The same. A Room in Shylock's House Capell.* Launcelot] *Rowe; om. Q, F.* 1. I am] *Q, F;* I'm *Pope.* 9. in] *Q; om. F.* 10. *Laun.*] *Q2; Clowne. Q, F.* 10. Adieu!] Adiew, *Q, F.* 11. Jew!] Iewe, *Q, F.* do] *Q, F;* did *F2.* 12. adieu!] adiew, *Q, F.* 13. something] *Q;* somewhat *F.* 14. adieu!] adiew. *Q, F.* S. D.] *Q2, F; om. Q; after l. 15 Capell.* 17. child!] child, *Q, F;* Child? *Rowe.*

5. *soon*] early.

10. *exhibit*] Eccles paraphrased "My tears serve to express what my tongue should, if sorrow would permit it," but probably it is Launcelot's blunder for prohibit (Halliwell) or inhibit (Clarendon).

10–11. *pagan*] This may have a scurrilous undertone: cf. *2 H 4,* II. ii. 168.

11. *do*] Malone upheld the reading of Qq and F by comparing II. vi. 23: "When you shall please to play the thieves for wives"; Launcelot seems fond of hinting at what is going to happen (cf. II. v. 22–3). If F2's "did" is accepted, *get* is used for beget, as in III. v. 9.

13–14. *foolish... spirit*] "tears do not become a man" (*AYL.,* III. iv. 3); cf. also *H 5,* IV. vi. 28–32.

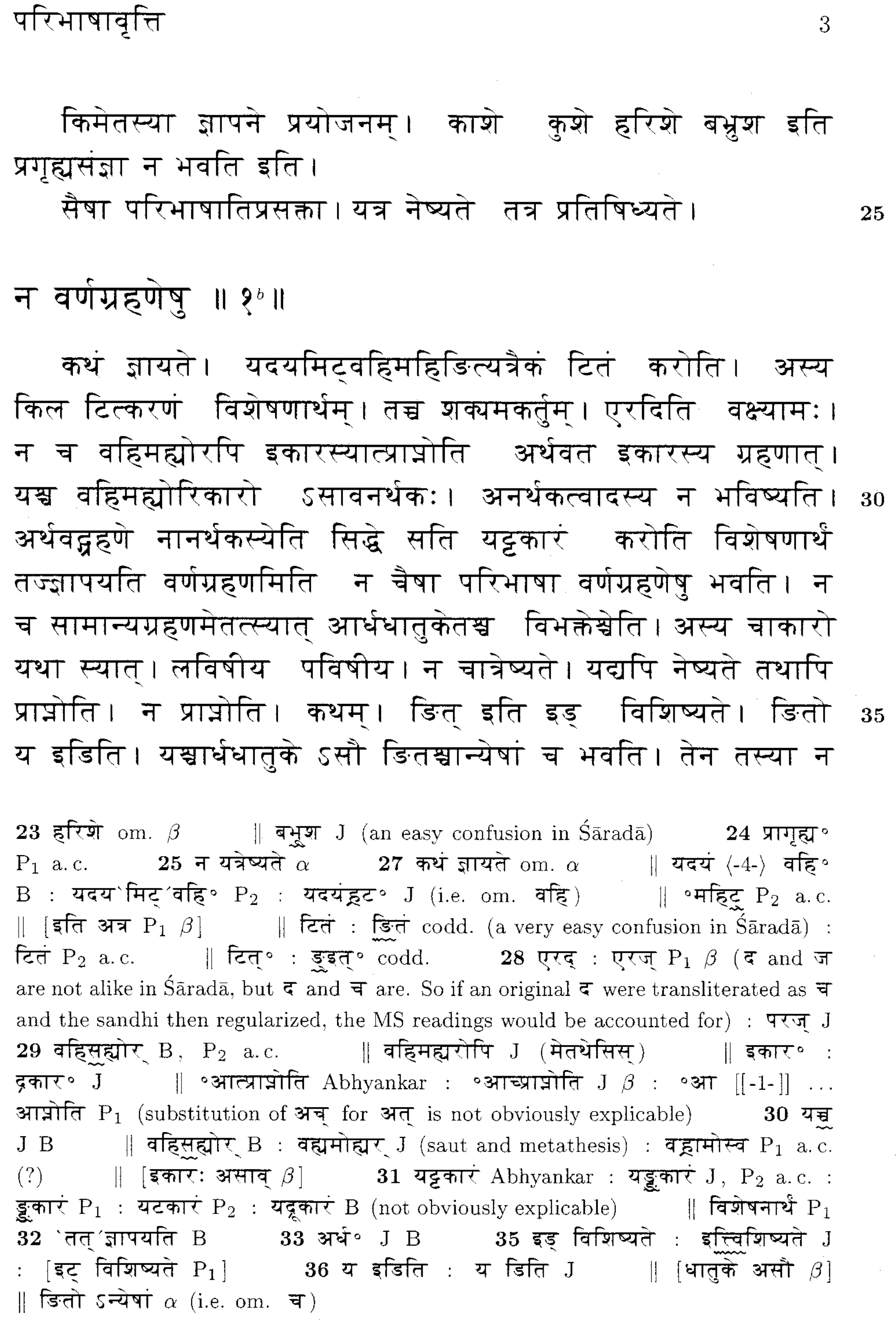
Figure 3: Output of the Arden text.

```
\font\sixsy=cmsy8 \skewchar\sixsy='60 \font\sixbf=cmbx8
\font\sixtt=cmtt8 \hyphenchar\sixtt=-1 % inhibit hyphenation
\font\sixsl=cmsl8 \font\sixit=cmti8
\def\eightpoint{\def\rm{\fam0\eightrm}%
  \textfont0=\eightrm \scriptfont0=\sixrm \scriptscriptfont0=\fiverm
  \textfont1=\eighti \scriptfont1=\sixi \scriptscriptfont1=\fivei
  \textfont2=\eightsy \scriptfont2=\sixsy \scriptscriptfont2=\fivesy
  \textfont3=\tenex \scriptfont3=\tenex \scriptscriptfont3=\tenex
  \def\it{\fam\itfam\eightit}\textfont\itfam=\eightit
  \def\sl{\fam\slfam\eightsl}\textfont\slfam=\eightsl
  \def\bf{\fam\bffam\eightbf}\textfont\bffam=\eightbf
  \scriptfont\bffam=\sixbf \scriptscriptfont\bffam=\fivebf
  \def\tt{\fam\ttfam\eighttt}\textfont\ttfam=\eighttt
  \normalbaselineskip=9pt   \global\let\sc=\fiverm
  \setbox\strutbox=\hbox{\vrule height7pt depth2pt width0pt}%
  \normalbaselines\rm}

% Macros for the edition:
\def\stage#1{\rlap{\hbox to \the\linenumsep{\hfil\llap{[{\it#1\/}]}}}}
\def\speaker#1{\pstart\hangindent2em\hangafter1
  \leavevmode{\it#1}\enspace\ignorespaces}
\def\\{\hfil\break}

% EDMAC customizations:
\noendnotes \vsize 40pc \hsize 23pc \parindent 0pt
\linenumsep=.3in \rightskip\linenumsep
\let\notenumfont=\eightrm  \let\notetextfont=\eightit \let\numlabfont=\eighti
\let\afootnoterule=\relax \let\bfootnoterule=\relax
\afootbaselineskip=9pt    \bfootbaselineskip=9pt
\footline={\hfil}
\def\rightlinenum{\numlabfont\llap{\the\line@num}}
\pageno=46
\headline={\eightpoint{\teni\folio}\hfil THE MERCHANT OF VENICE\hfil [ACT II}
\cropsetup{8in}{5in}{3.5pc}{3pc}
\hoffset=.75in   \voffset=.9375in
\frenchspacing

% Footnote formats:
\def\nonumparafootfmt#1#2#3{%  footnote format that doesn't have line numbers
  \let\par=\endgraf
  \rightskip=0pt \leftskip=0pt
  \parindent=0pt \parfillskip=0pt plus 1fil
  {\eightpoint\lemmafont#1|#2\/\rm]\enskip\notetextfont
       #3\penalty-10\hskip 1em plus.5em minus.1em\relax}}
\def\newparafootfmt#1#2#3{%
  \let\par=\endgraf
  \rightskip=0pt \leftskip=0pt
  \parindent=0pt \parfillskip=0pt plus 1fil
  {\eightpoint\notenumfont\printlines#1|\rm.\enspace
      \lemmafont#1|#2\/\rm]\enskip\notetextfont
       #3\penalty-10\hskip 1em plus.5em minus.1em\relax}}
\def\newtwocolfootfmt#1#2#3{%
  \let\par=\endgraf
  \hsize .48\hsize
  \rightskip=0pt \leftskip=0pt \parindent=5pt
  {\eightpoint\notenumfont\strut\printlines#1|\rm.\enspace
      \it#2\/\rm]\penalty100\hskip .5em plus .5em\rm
       #3\strut\endgraf\allowbreak\relax}}
```

```
% Footnote style selections etc. (done last):
\footparagraph{a}
\foottwocol{b}
\let\afootfmt=\newparafootfmt
\let\bfootfmt=\newtwocolfootfmt
\let\collation=\afootnote
\let\note=\bfootnote
\lineation{section}
\linenummargin{right}
\makeatother
```

The Arden text, using the above definitions, is input as follows (the output is shown in Figure 3):

```
\input edmac
\input arden.sty

\let\afootfmt=\nonumparafootfmt % we do not want line numbers initially

\beginnumbering
\pstart
\centerline{[\text{SCENE III}
  \lemma{Scene III}
  \collation{Capell; om. Q, F; {\rm Scene IV} Pope.}/.---%
  \text{\it Venice}
  \collation{om. Q, F; Shylock's house Theobald; The same.
  A Room in Shylock's House Capell.}/.]}
\pend
\bigskip

\pstart
\centerline{\it Enter\/ {\rm JESSICA} and\/
  {\rm [\text{LAUNCELOT}
  \lemma{Launcelot}
  \collation{Rowe; om. Q, F.}/]} the clown.} \pend \bigskip

\let\afootfmt=\newparafootfmt % we do want line numbers from now

\speaker{Jes.} \setline{1}%
\text{I am}
  \collation{Q, F; {\rm I'm} Pope.}/
sorry thou wilt leave my father so,\\
Our house is hell, and thou (a merry devil)\\
Didst rob it of some taste of tediousness,---\\
But fare thee well, there is a ducat for thee,\\
And Launcelot, \text{soon}
  \note{early.}/
at supper shalt thou see\\
Lorenzo, who is thy new master's guest,\\
Give him this letter,---do it secretly,---\\
And so farewell: I would not have my father\\
See me \text{in}
  \collation{Q; om. F.}/
talk with thee.
\pend

\speaker{Laun.}
```



परिभाषावृत्ति 3

किमेतस्या ज्ञापने प्रयोजनम्। काशे कुशे हरिशे बभ्रुश इति प्रगृह्यसंज्ञा न भवति इति।
सैषा परिभाषातिप्रसक्ता। यत्र नेष्यते तत्र प्रतिषिध्यते।

न वर्णग्रहणेषु ॥ १^{b}॥

कथं ज्ञायते। यदयमिट्वहिमहिङित्यत्रैकं टितं करोति। अस्य किल टित्करणं विशेषणार्थम्। तच्च शक्यमकर्तुम्। एरदिति वक्ष्यामः। न च वहिमह्योरपि इकारस्यात्प्राप्नोति अर्थवत इकारस्य ग्रहणात्। यश्च वहिमह्योरिकारो ऽसावनर्थकः। अनर्थकत्वादस्य न भविष्यति। अर्थवद्ग्रहणे नानर्थकस्येति सिद्धे सति यट्टकारं करोति विशेषणार्थं तज्ज्ञापयति वर्णग्रहणमिति न चैषा परिभाषा वर्णग्रहणेषु भवति। न च सामान्यग्रहणमेतत्स्यात् आर्धधातुकेतश्च विभक्तेश्चेति। अस्य चाकारो यथा स्यात्। लविषीय पविषीय। न चात्रेष्यते। यद्यपि नेष्यते तथापि प्राप्नोति। न प्राप्नोति। कथम्। ङित् इति इङ् विशिष्यते। ङितो य इडिति। यश्चार्धधातुके ऽसौ ङितश्चान्येषां च भवति। तेन तस्या न

23 हरिशे om. β ‖ बभूश J (an easy confusion in Śāradā) **24** प्रागृह्य° P$_1$ a.c. **25** न यत्रेष्यते α **27** कथं ज्ञायते om. α ‖ यदयं ⟨-4-⟩ वहि° B : यदय`मिट्´वहि° P$_2$: यदयंह्रट° J (i.e. om. वहि) ‖ °महिट्‌ P$_2$ a.c. ‖ [इति अत्र P$_1$ β] ‖ टितं : ङितं codd. (a very easy confusion in Śāradā) : टितं P$_2$ a.c. ‖ टित्° : ङ्इत्° codd. **28** एरद् : एरज् P$_1$ β (द and ज are not alike in Śāradā, but द and च are. So if an original द were transliterated as च and the sandhi then regularized, the MS readings would be accounted for) : परज् J **29** वहिसह्योर् B, P$_2$ a.c. ‖ वहिमह्यरोपि J (मेतथेसिस्) ‖ इकार° : द्रकार° J ‖ °आत्प्राप्नोति Abhyankar : °आच्प्राप्नोति J β : °आ [[-1-]] … आप्नोति P$_1$ (substitution of अच् for अत् is not obviously explicable) **30** यच्च J B ‖ वहिसह्योर् B : वह्यमोह्यर् J (saut and metathesis) : वह्रामोस्व P$_1$ a.c. (?) ‖ [इकारः असाव् β] **31** यट्टकारं Abhyankar : यड्डकारं J, P$_2$ a.c. : ड्डकारं P$_1$: यटकारं P$_2$: यदूकारं B (not obviously explicable) ‖ विशेषनार्थं P$_1$ **32** `तत्´ज्ञापयति B **33** अर्ध° J B **35** इङ् विशिष्यते : इत्त्विशिष्यते J : [इट् विशिष्यते P$_1$] **36** य इडिति : य डिति J ‖ [धातुके असौ β] ‖ ङितो ऽन्येषां α (i.e. om. च)

Figure 4: Sanskrit edition of a grammatical text.

```
\text{}\lemma{\it Laun.}
  \collation{Q2; Clowne. Q, F.}/%
\text{Adieu!}
  \collation{{\rm Adiew}, Q, F.}/
tears \text{exhibit}
  \note{Eccles paraphrased ``My tears serve to express what my
  tongue should, if sorrow would permit it,'' but probably it is
  Launce\-lot's blunder for prohibit (Halliwell) or inhibit
  (Clarendon).}/
my tongue, most beautiful
\text{pagan}
  \note{This may havt a scurrilous undertone: cf. {\it 2 H 4,}
  \sc II. \rm ii. 168.}/,
most sweet \text{Jew!}
  \collation{{\rm Iewe}, Q, F. \quad {\rm do]} Q, F; {\rm did} F2.}/---if
a Christian \text{do}
  \note{Malone upheld the reading of Qq and F by comparing \sc
  II. \rm vi. 23: ``When you shall please to play the thieves for
  wives''; Launcelot seems fond of hinting at what is going to
  happen (cf. \sc II. \rm v. 22--3). If F2's ``did'' is accepted,
  {\it get\/} is used for beget, as in \sc III. \rm v. 9.}/
not play
the knave and get thee, I am much deceived; but
\text{adieu!}
  \collation{{\rm adiew}, Q, F.}/
these \text{foolish drops do \text{something}
  \collation{Q; {\rm somewhat} F.}/
drown my
manly spirit}
  \lemma{foolish{\rm\dots}spirit}
  \note{``tears do not become a man'' (\it AYL., \sc III. \rm
  iv. 3); cf. also \it H 5, \sc IV. \rm vi. 28--32.}/:
\text{adieu!}
  \collation{{\rm adiew}. Q, F. \quad {\rm S. D.]} Q2, F; om. Q; after
  l. 15 Capell.}/
\hfill \stage{Exit.}
\pend

\speaker{Jes.}
Farewell good Launcelot.\\
Alack, what heinous sin is it in me\\
To be ashamed to be my father's \text{child!}
  \collation{{\rm child}, Q, F; {\rm Child?} Rowe.}/
\pend
\endnumbering
\bye
```

3.3 Sanskrit text edition

Finally, Figure 4 shows an example from an edition of a Sanskrit text on Pāṇinian grammar that uses Frans Velthuis's excellent Devanāgarī font. I have not shown the input file for this because I almost never looked at it myself. The edition records a large number of variants, and there are frequent font and script changes. Preparing this purely manually would have been very error-prone. In fact, the text was prepared using a word processor which had the ability to fold footnotes out of sight. I designed custom Indic fonts for my computer screen, so that I could see all the diacritical marks on accented characters as I typed. (A set of TEX macros declared these characters active, and defined them to give the correct output.) Font changes were invoked using the standard facilities of the word processor, so the perennial "missing closing brace" hardly ever arose. A short post-processor program changed the word processor file into correctly tagged `EDMAC` input, and another post-processor (provided by Velthuis) did some special processing on the Devanāgarī strings. This combination of tools proved very workable and no major problems were encountered.

4 Index

All numbers denote pages where the corresponding entry is discussed.

◇ John Lavagnino
Department of English and American Literature
Brandeis University
415 South Street
Waltham, MA 02254–9110, USA
Bitnet: `lav@brandeis`
Internet: `lav@binah.cc.brandeis.edu`

◇ Dominik Wujastyk
Wellcome Institute for the History of Medicine
183 Euston Road
London NW1 2BN, UK
Janet: `D.Wujastyk@uk.ac.ucl`
Internet: `dow@wjh12.harvard.edu`

LaTeX

Footnote Strikes A Wrong Chord: How Do You Conquer It?

Jackie Damrau

Have you ever tried to change the footnote symbol? Did you find it easy? Well, the way to change the type of symbol used by the `\footnote` environment is, e.g., by:

```
\renewcommand{\thefootnote}
             {\fnsymbol{footnote}}
```

which provides the $*$, $**$, †, ‡, §, ¶, @ construction. Should you desire alphabetic (upper or lowercase), the constructions would be:

```
\renewcommand{\thefootnote}{\Alph{footnote}}
\renewcommand{\thefootnote}{\alph{footnote}}
```

or could conceivably be anything else you would wish to use.

Query from the College Station Meeting

Lowell Smith posed a question at the 11th Annual TeX Users Group Meeting in College Station where he wanted to change the `\footnote` environment inside a minipage and no matter what he tried received lowercase alphabet letters for the footnote symbol.

I would like to throw this question out to the LaTeX community to see what responses are submitted. Any solutions submitted will be reviewed in the next column.

Earlier Column Revisited

In *TUGboat* 11, no. 1, this column presented macros for changing from single space to double space. I received a response from Josephine Colmenares at Fordham University with a simpler macro that she has given permission to use. The macros are:

```
\newcommand{\single}{%
  \renewcommand{\baselinestretch}{1}
  \normalsize}
\newcommand{\double}{%
  \renewcommand{\baselinestretch}{1.5}
  \normalsize}
```

◇ Jackie Damrau
Computer Operations Support
MS-1011
SSC Laboratory
2550 Beckleymeade Avenue
Dallas, TX 75237-3946
(214) 708-6048
Internet: damrau@sscvx1.ssc.gov

A LaTeX Document Style Option for Typesetting *APL*

Andreas Geyer-Schulz, Josef Matulka and Gustaf Neumann

Abstract

In this article we describe the LaTeX document style option `apl.sty` for typesetting documents containing passages in *APL* code. All symbols needed within a multi-vendor *APL* environment are provided. Currently the full symbol sets of the *APL* dialects *APL*2, Dyalog *APL*, I-*APL*, Sharp *APL*, and *APL*.68000 are supported. All *APL* symbols are constructed with the symbols of the standard LaTeX font family. No additional fonts are needed. Standard LaTeX commands can be used to change the size and type style of *APL* symbols. Automatic conversion of *APL* objects is achieved by a preprocessor written in *APL*. LaTeX, `apl.sty` and the preprocessor bundled together provide an integrated high-quality *APL* publishing system.

1 Motivation

Why do you insist on using a notation which is a nightmare for typist and compositor and impossible to implement with punching and printing equipment currently available?
– R. A. Brooker, 1963 [Iverson 63]

Since its introduction in the early sixties, *APL* has been known and even become famous not so much for the power and elegance of its concepts but — to a much greater extent — for the "strange" symbols it uses. A discussion about the usefulness and difficulties of *APL* has remained academic for large parts of the computer science community. Many programmers never managed to get their dumb ASCII–terminals to produce the non-ASCII symbols required by the language.

With the advent of bit-mapped displays, downloadable fonts and the spread of graphical interfaces such as the X Window System the situation has changed. Specialized hardware is no longer a prerequisite for *APL* programming. Although it often requires some effort of customization and configuration, it is possible nowadays to turn existing hardware into an *APL* environment.

Troubles show up as soon as you start publishing results produced in your *APL* environment. (Just have a look at some books on *APL*, where the *APL* passages had to be pasted in!) Since many text-processing and desktop-publishing systems still lack *APL* support, it remains difficult to achieve high printing quality in publications composed of text and *APL* code. Several extensions to existing text processors have been implemented (cf. [Hohti, Kanerva 88]). However, most of them support either only the symbols of one *APL* dialect or only one machine or operating system platform. [Hohti, Kanerva 88] already demonstrated the usefulness of TEX for *APL* typesetting. The authors produced a METAFONT description for *APL* primitive symbols and a set of TEX macros to support Digital's *APL* interpreter for the VAX-11 series.

In this paper we present our solution to the problem: An *APL* publishing system consisting of an *APL* front end and a LATEX document style option. The *APL* front end automatically converts *APL* material into LATEX code which you can `\input` into any standard LATEX document. The LATEX document style option `apl.sty` provides macros defining all *APL* characters as combinations of standard LATEX symbols, thus relieving us from the burden of designing new fonts and the user from the task of incorporating them into the LATEX system. As additional benefit, size and type style of the *APL* symbols can be changed by the familiar LATEX commands (e.g. `\Large`, `\sf`).

Compared with the approach of Hohti and Kanerva mentioned above, our solution offers the following advantages:

- Several *APL* dialects are supported (currently *APL*2, Dyalog *APL*, I-*APL*, Sharp *APL*, and *APL*.68000).
- No additional fonts are required.
- *APL* symbols for all LATEX sizes and type styles are available.
- We provide support for automatic typesetting of *APL* objects.

There are some disadvantages, however; they are higher TEX interpretation overhead and higher TEX memory usage.

2 The *APL* Publishing System

The *APL* publishing system consists of two parts, the *APL* front end and the LATEX document style option `apl.sty` which communicate via a carefully designed interface of TEX macros (see Figure 1). This ensures that both parts of the system can be modified independently.

Each of these modules is composed of two layers (see Table 1). The main task of the *low level formatting* layer is the printing of single *APL* symbols. The *APL* front end maps each symbol into a TEX macro name and produces files to be `\input` into LATEX documents. The LATEX style option `apl.sty` contains one macro definition for each *APL* character.

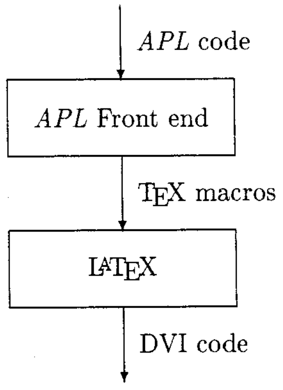


Figure 1: Modules of the Publishing System

APL code is more than just a stream of *APL* symbols. The *high level formatting* layer knows about functions, operators, arrays, and expressions. Our *APL* front end provides special functions for typesetting these objects. The LATEX style option defines the corresponding environments.

	APL front end	LATEX document style option
Low level formatting	symbol translation	symbol construction
High level formatting	*APL* language elements	logical document elements

Table 1: Layers of the Publishing System

3 Typesetting *APL* Symbols

If using a few *APL* symbols in an ordinary document is all you need, you can forget about the *APL* front end. Simply adding the option `apl` to your preferred LATEX document style (e.g. `\documentstyle[12pt,apl]{article}`) enables you to state in your paper, e.g.:

> By combining the simple *APL* symbols ○ and ⋆ we obtain the compound symbol ⍟.

The code to produce this statement is:

```
\begin{quotation}
By combining the simple \APL\ symbols
\APLcircle\ and \APLstar\ we obtain the
compound symbol \APLcirclestar.
\end{quotation}
```

In fact, you can typeset all simple and compound symbols of *APL2*, as we have defined macros for all of them. Tables 2 and 3, respectively, show them together with their macro names.

\APLalpha	α	\APLnotgreater	≤
\APLbar	−	\APLnotless	≥
\APLcircle	○	\APLomega	⍵
\APLcolon	:	\APLoverbar	¯
\APLcomma	,	\APLplus	+
\APLdel	∇	\APLquad	⎕
\APLdelta	∆	\APLquery	?
\APLdieresis	¨	\APLquote	'
\APLdivide	÷	\APLrho	⍴
\APLdot	.	\APLrightarrow	→
\APLdownarrow	↓	\APLrightbracket	]
\APLdowncaret	∨	\APLrightparen	)
\APLdownshoe	∪	\APLrightshoe	⊃
\APLdownstile	⌊	\APLsemicolon	;
\APLdowntack	⊥	\APLslash	/
\APLepsilon	∊	\APLslope	\
\APLequal	=	\APLstar	⋆
\APLgreater	>	\APLstile	∣
\APLiota	⍳	\APLtilde	∼
\APLjot	∘	\APLtimes	×
\APLleftarrow	←	\APLunderbar	_
\APLleftbracket	[	\APLuparrow	↑
\APLleftparen	(	\APLupcaret	∧
\APLleftshoe	⊂	\APLupshoe	∩
\APLless	<	\APLupstile	⌈
\APLmathbar	−	\APLuptack	⊤
\APLnotequal	≠		

Table 2: Simple *APL2* Symbols

\APLcirclebar	⊖
\APLcircleslope	⍉
\APLcirclestar	⍟
\APLcirclestile	⌽
\APLdelstile	⍒
\APLdeltastile	⍋
\APLdeltaunderbar	⍙
\APLdeltilde	⍫
\APLdieresisdot	∵
\APLdowncarettilde	⍱
\APLdowntackjot	⍎
\APLdowntackuptack	⌶
\APLepsilonunderbar	⍷
\APLequalunderbar	≡
\APLiotaunderbar	⍸
\APLleftbracketrightbracket	⌷
\APLquaddivide	⌹
\APLquadjot	⌻
\APLquadquote	⍞
\APLquadslope	⍂
\APLquotedot	!
\APLslashbar	⌿
\APLslopebar	⍀
\APLupcarettilde	⍲
\APLupshoejot	⍝
\APLuptackjot	⍕

Table 3: Compound *APL2* Symbols

IBM was the first company to implement *APL* but it did not remain the only one. Companies such as I. P. Sharp and Dyadic Systems have produced their own versions of the language. These and other companies, however, introduced only a few symbols not found in *APL2*. We have added twenty additional symbols to the *APL2* character set to support typesetting Dyalog *APL*, I-*APL*, Sharp *APL*, and *APL*.68000 (see Table 4).

As you have probably guessed from the names in Tables 2, 3, and 4 we stick to a naming convention in order to minimize name clashes with other macro packages and also help users remembering the macro names. All macro names start with the \APL prefix, followed by the name of the symbol used in the *APL* literature. The symbol names for *APL2* characters are taken from [IBM 85]. For those characters (cf. Table 4) which are not included in the IBM list we have invented consistent names. We always use symbol names, not the name of *APL* functions these symbols might represent. The name of a compound *APL* symbol is the concatenation of the names of the simple *APL* symbols it is created from.

As can be seen in Figure 2 which shows the character set (the atomic vector ⎕AV) of *APL2*, not all *APL* characters are fancy symbols, and the language uses ordinary alphanumeric characters as well. To allow for a clean interface between the *APL* front end and the LaTeX part of our system, we decided to define macros for these characters as well. Their names are constructed as follows:

- Each macro name starts with \APL.
- For each letter we append the upper or lowercase letter, if the letter is underlined we prefix the letter with "u".
  Capital letters: \APLA, ..., \APLZ.

`\APLOUT`	⦶
`\APLasciipound`	£
`\APLbarcomma`	⍪
`\APLcaret`	^
`\APLdiamond`	⋄
`\APLdieresiscircle`	⍥
`\APLdieresisdel`	⍢
`\APLdieresisjot`	⍤
`\APLdieresisstar`	⍣
`\APLdieresistilde`	⍨
`\APLdieresisuptack`	⍡
`\APLlefttack`	⊢
`\APLnotequalunderbar`	≠̲
`\APLquaddownarrow`	⍗
`\APLquadleftarrow`	⍇
`\APLquadrightarrow`	⍈
`\APLquaduparrow`	⍐
`\APLrighttack`	⊣
`\APLstilebar`	⍭
`\APLtheta`	θ

Table 4: Symbols Used in APL Dialects

Lowercase letters: `\APLa`, ..., `\APLz`.
Underlined capital letters: `\APLuA`, ..., `\APLuZ`.
Underlined lowercase letters: `\APLua`, ..., `\APLuz`.

- For numbers we simply append their names: `\APLzero`, ..., `\APLnine`.

The tiny numbers in the atomic vector of Figure 2 correspond to positions for which no printable characters are defined by *APL2*. In case the *APL* front end encounters a nonprintable character, e.g. the one at position 20 in ⎕AV, it generates `\APLmiss{20}`. The definition of the LaTeX macro `\APLmiss` determines the printed representation of this character (the default macro in our style just prints the corresponding number in style `\tiny`).

Let us close this section with one more example of typesetting *APL* symbols:

```
\APLquaddivide\APLA\ corresponds to
$A^{-1}$ in mathematical notation
and \APLcircleslope\APLA\ corresponds
to $A^{T}$.
```

displays as:

⌹A corresponds to A^{-1} in mathematical notation and ⍉A corresponds to A^T.

0	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	31
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
	A̲	B̲	C̲	D̲	E̲	F̲	G̲	H̲	I̲	¢	.	<	(	+	|
&	J̲	K̲	L̲	M̲	N̲	O̲	P̲	Q̲	R̲	!	$	⋆	)	;	¬
−	/	S̲	T̲	U̲	V̲	W̲	X̲	Y̲	Z̲	¦	,	%	_	>	?
112	∧	¨	⌻	⍸	⍷	118	119	∨	‘	:	#	@	'	=	”
~	a	b	c	d	e	f	g	h	i	↑	↓	≤	⌈	⌊	→
⎕	j	k	l	m	n	o	p	q	r	⊃	⊂	156	○	158	←
¯	~	s	t	u	v	w	x	y	z	∩	∪	⊥	[	≥	∘
α	∊	⍳	⍴	⍵	181	×	\	÷	185	∇	∆	⊤	]	≠	|
{	A	B	C	D	E	F	G	H	I	⍲	⍱	⌷	⌽	⍂	⍉
}	J	K	L	M	N	O	P	Q	R	⌶	!	⍒	⍋	⍞	⍝
\	≡	S	T	U	V	W	X	Y	Z	⌿	⍀	⍨	⊖	⌹	⍕
0	1	2	3	4	5	6	7	8	9	250	⍫	⍙	⍟	⍎	255

Figure 2: The Atomic Vector of *APL2*

4 Typesetting *APL* Objects

Typing the name of an occasional *APL* symbol within a normal text is not a real nuisance to the author of *APL* texts. But typesetting a larger piece of *APL* code certainly is. Imagine a function named ∆TREE, which implements a recursive tree traversal algorithm:

```
    ∇Z←CLASS_LIST ∆TREE ROOT;
     DEPTH;LIST;I;RECLIST;SUPERCLASS
[1]  Z←,⊂ROOT
[2]  →(∨/(,⊂ROOT)≡¨CLASS_LIST)/
     CYCLIC
[3]  I←2 ⎕TF 2⊃GET_MEM '∆CLASS'(
     ROOT,'.SUPERCLASS')
[4]  →((0⍴⊂1⍴' ')≡SUPERCLASS)/0
[5]  I←(⊂CLASS_LIST,⊂ROOT)∆TREE¨
     SUPERCLASS
[6]  Z←((Z⍳Z)=⍳⍴Z)/Z←Z,↑,/,¨,I
[7]  →0
[8]  CYCLIC:Z←0⍴0
    ∇
```

In order to print just the beginning of the header of the function, you would have to type:

```
\APLspace\APLspace\APLspace\APLspace%
\APLspace\APLdel\APLZ%
\APLleftarrow\APLC\APLL\APLA\APLS\APLS%
\protect\APLunderbar\APLL%
\APLI\APLS\APLT\APLspace\protect\APLdelta%
```

```
\APLT\APLR\APLE\APLE%
\APLbr\APLspace\APLR\APLO\APLO\APLT\APLbr%
```

Obtaining the familiar function layout used in *APL* textbooks would require additional code. What is more, besides being awkward the whole process is error-prone: Almost certainly it will result in a printout different from the *APL* code.

Therefore we strongly recommend automatic translation of *APL* code. We provide an *APL* front end which transforms *APL* objects into logical document elements which can be `\input` into LaTeX documents. This guarantees consonance between the original *APL* code and its listing and is also more convenient.

For all *APL* language elements we have defined *APL* functions and corresponding LaTeX environments. Our system supports the typesetting of:

- an array displayed by the interpreter,
- an array in boxed representation,
- a function or operator displayed by the built in *APL* del-editor (∇-editor),
- a function or operator displayed by *APL*'s canonical representation function ⎕CR,
- a direct definition of a function or operator,
- an *APL* expression input by the user.

Apart from minor modifications we have used traditional layout conventions for all language elements. For example, the convention that user input is six spaces indented can be traced back to the very first implementation of *APL*. Another traditional convention states that if a line of *APL* code does not fit on a single line of the display, the rest of the code is wrapped around and continues on the next line. In some functions this rule may lead to line breaks in the middle of names. Since *APL* identifiers can be up to 255 characters long line breaking within names cannot be avoided in general.

As you can see, arrays and functions can be typeset in various ways. For example, the above listing of the *APL* function ∆TREE was printed by the following *APL* expression:

'TREE' PRTEX_FN '∆TREE'

The *APL* function PRTEX_FN produces the file `tree.tex` as output. The *APL* front end not only maps each character into the corresponding TeX macro but it also produces the line numbers in brackets and the surrounding LaTeX environments in order to guarantee uniform display of functions throughout the document.

In the following we present examples for each of the cases mentioned above. At the same time, the examples give us the opportunity to demonstrate variations of type style and size.

4.1 Typesetting *APL* Arrays

The interpreter usually displays arrays as text matrices on the screen. For example, the matrix X is displayed as:

```
 1  2      A
 3  4

 B         C
```

The above printout is typeset by the following code which is automatically produced by the *APL* front end:

```
% AR1 X
\begin{APLarray}
\APLmb{\APLspace}\APLmb{\APLone}
\APLmb{\APLspace}\APLmb{\APLtwo}
\APLmb{\APLspace}\APLmb{\APLspace}
\APLmb{\APLspace}\APLmb{\APLA}
\APLspace\par
   ...
\end{APLarray}
```

Note that the structure of X has been preserved by automatically enforcing fixed spacing. A closer examination of the code reveals that we have simulated fixed spacing by boxing each character of the array (`\APLmb` does this).

Experienced *APL* programmers recognize the structure of X at the first glance: X is a two by two matrix whose upper left element is a two by two matrix. However, since the use of nested arrays is typical for second generation *APL*s like *APL2* and Dyalog *APL*, another representation of arrays exists which shows the structure in a more explicit manner:

```
.→- - - - - - - - - .
↓  .→- - .          |
|  ↓ 1  2 |    A    |
|  | 3  4 |    -    |
|  '∼- - '          |
|                   |
|    B         C    |
|    -         -    |
'∊- - - - - - - - - '
```

Most of the work for typesetting the boxed representation of X shown above is done by the *APL* function DISPLAY which usually comes with the *APL* system (e.g. [IBM 85]). Our *APL* front end just translates the characters generated by this function; the same LaTeX environment is used for both array representations. We only sketch the code for the boxed representation:

```
{\it
\begin{APLarray}
\APLmb{\APLdot}\APLmb{\APLrightarrow}
\APLmb{\APLbar}\APLmb{\APLbar}
  ...
\end{APLarray}
}
```

In order to demonstrate the ease of changing type styles we have decided to put the generated code unit into an *italics* environment. This is the reason for all letters and numbers in the boxed representation being in *italics*. Otherwise, they would have been roman.

4.2 Typesetting *APL* Functions

The following *APL* function is printed in del-editor style:

```
     ∇Z←MEAN X
[1]    Z←(+/X)÷ρX
     ∇
```

A larger font has been selected by inserting the generated code into a \Large environment:

```
\begin{Large}
% FNS MEAN
\begin{APLfns}
\begin{APLfnsline}{}{\APLdel}
\APLZ\APLleftarrow\APLM\APLE\APLA\APLN
\APLspace\APLX
\end{APLfnsline}
\begin{APLfnsline}{\APLleftbracket\APLone
\APLrightbracket}{}
  ...
\end{APLfns}
\end{Large}
```

The canonical representation of an *APL* function is simply a text matrix. Since older *APL* systems only provide arrays of uniform datatype and rectangular shape, padding of short lines with spaces is performed. In contrast to the del-editor style, the canonical representation is typeset like an *APL* array with fixed spacing and without line numbering.

For the canonical representation of MEAN a small typewriter type style was chosen:

```
Z←M E A N   X
Z←( + / X ) ÷ ρ X
```

You notice immediately that we have used the LaTeX commands \small and \tt to produce this effect:

```
{\small\tt
% CR MEAN
\begin{APLcr}
\APLmb{\APLZ}\APLmb{\APLleftarrow}
\APLmb{\APLM}\APLmb{\APLE}
\APLmb{\APLA}\APLmb{\APLN}
\APLmb{\APLspace}\APLmb{\APLX}
\APLspace\par
  ...
\end{APLcr}
}
```

In addition to the del-editor representation and the canonical representation of an *APL* function we provide means for formatting direct definitions of functions, which are supported by only a few *APL* dialects, e.g. I-*APL*:

fib: z,+/⁻2↑z←fib ω−1 : ω=1 : 1

This direct definition of a function computing Fibonacci numbers is due to [Iverson 87] and has been formatted as follows:

```
\begin{APLline}
\APLf\APLi\APLb\APLcolon\APLspace\APLz
\APLcomma\APLplus\APLslash
\APLoverbar\APLtwo\APLuparrow\APLz
  ...
\end{APLline}
```

Note, that the APLline environment allows ligatures within names.

4.3 Typesetting *APL* Expressions

Finally, our system enables the user to typeset *APL* expressions. The expression

X←2 2ρ(2 2ρι4) 'A' 'B' 'C'

which happens to be the one used to generate the matrix *X* (our example for formatting arrays) is printed by:

```
% EXPR
\begin{APLbold}\begin{APLexpr}
\APLX\APLleftarrow\APLtwo\APLspace
```

```
\APLtwo\APLrho
\APLleftparen\APLtwo\APLspace\APLtwo
  ...
\end{APLexpr}\end{APLbold}
```

The environment `APLexpr` provides the traditional six space indentation for user input.

4.4 Typesetting User Dialogues and Workspaces

The above examples demonstrate the usefulness of the six basic document elements we provide. Besides being useful on their own, we can combine them to form higher level units. In the current version of the *APL* front end, support for typesetting a dialogue and an *APL* workspace listing is included.

A dialogue is a pair of user input and interpreter response. We typeset the user input as an *APL* expression and the interpreter response as an *APL* array. The dialogue

```
      (1(2 3)4
SYNTAX  ERROR
          ( 1 ( 2   3 ) 4
          ∧
```

was generated by entering the following *APL* expression

'E' PRTEX_DIALOG '(1(2 3)4'

Hopefully, most of the dialogues will not result in a syntax error as the above one. But automatically typesetting examples with errors is very convenient for describing *APL*'s error trapping mechanisms in a text book.

An example for the printout of an *APL* workspace would be too space-consuming to be included here. It basically is implemented by combining the printout of arrays, functions and operators intertwined with LaTeX sectioning commands. LaTeX's table of contents considerably increases the utility of a workspace listing.

5 Implementation Details

For symbol construction three internal macros had to be defined. The first, `\@APLmath`, puts a math symbol into a boxed math environment and adjusts spacing. The second, `\@APLmraise`, puts a math symbol into a raised and boxed math environment and adjusts spacing. The third, `\@APLovly`, simulates backspacing and overstriking on a typewriter by overlaying two boxes. The quad symbol ⎕ required special construction in order to generate readable compound symbols such as ⍂, ⌹, ⍈. Full reconstruction was needed only for the *APL* symbol ⍞ which is a quad symbol with a short vertical rule. The first version of this symbol used a single quote instead of the vertical rule and looked rather awkward.

One disadvantage of our solution is high TeX memory consumption. We have used `\let` commands wherever possible in order to cut down memory usage. Typesetting *APL* symbols for this text has cost us approximately 6,800 words of TeX memory. Typesetting the workspace of the *APL* front end (32 functions, 11 variables, 23 pages) has cost a total of 72,800 words of TeX memory with 29,000 words used for the *APL* symbols. We recommend TeX with 262,141 words of memory.

APL lines are sometimes too long to be printed in a single line. When displaying them on the screen, this problem is usually resolved by wrapping them to the next lines without adding any hyphen. Thus, line breaks can occur anywhere in an *APL* line, in the middle of *APL* expressions or even in names. As we use one macro for typesetting each *APL* character, the normal TeX hyphenation algorithm no longer works.

In the definition of each *APL* symbol which cannot be used in an identifier the macro `\APLgb` is used, which allows breaks with a penalty of -10. To achieve line breaks in the emergency case the preprocessor inserts the macro `\APLbr` into names longer than 15 characters at regular intervals.

When typesetting *APL* arrays (cf. Figure 2), a fixed spaced font is necessary to preserve its shape. We imitate fixed spaced fonts by simply putting a box of fixed width around all characters.

For typesetting bold *APL* code the special environment `APLbold` is defined. It sets `\bf` and `\boldmath` and adjusts the thickness of rules used in symbol construction.

6 Conclusion

In this paper we have presented our solution to the *APL* typesetting problem: An *APL* publishing system consisting of a LaTeX document style option and an *APL* front end. No additional fonts are needed. We have given short examples which demonstrate the usefulness of this approach. The system has already been used by the authors to prepare several articles published in *APL* Quote Quad, the journal of the *APL* user community.

METAFONT could be used to improve the printing quality of some symbols (cf. [Hohti, Kanerva 88]). However, it would be necessary to create a whole *APL* font family (different sizes and type styles) to obtain the flexibility of our system. We could incorporate special *APL* fonts without any change in the *APL* front end as soon as they become available.

The *APL* front end is currently implemented for *APL*2 and Dyalog *APL* and can be obtained from the authors. Further porting is intended. The LaTeX document style option will be submitted to the Clarkson and Aston archives as well as to the German TeX server at Heidelberg.

The authors would appreciate comments and suggestions for the improvement of the style as well as comments with regard to *APL* symbols not available in this style.

References

[Camacho et al. 87] Camacho A., Chapman P., Ziemann D. (1987), *I-APL Instruction Manual for PC Clones*, I-APL Limited, St. Albans, Herts, England.

[Dyadic Systems Ltd. 85] Dyadic Systems Limited (1985), *Lynwood Dyalog APL User Guide*, Dyadic Systems Limited, Farnborough, Hampshire, England.

[Falkoff, Iverson 73] Falkoff A. D., Iverson K. E. (1973) "The Design of APL", *IBM Journal of Research and Development*, V17, N4, reprinted in: Falkoff A. D., Iverson K. E. (1981), *A Source Book in APL*, APL Press, Palo Alto.

[Hohti, Kanerva 88] Hohti A., Kanerva O. (1988), "Typesetting APL with TeX", *APL Quote Quad*, V18, N3, p13–16.

[IBM 85] IBM Corporation (1985), *APL2 Programming: Language Reference*, IBM Corporation, San Jose, California.

[I.P. Sharp 85] I.P. Sharp Associates Limited (1985), *SHARP APL/PC Handbook*, I.P. Sharp Associates Limited, Toronto, Canada.

[Iverson 63] Iverson K. E. (1963), "Formalism in Programming Languages", ACM Working Conference on Mechanical Language Structures, Princeton N.J., reprinted in: Falkoff A. D., Iverson K. E. (1981), *A Source Book in APL*, APL Press, Palo Alto.

[Iverson 87] Iverson K. E. (1987), "A Dictionary of APL", *APL Quote Quad*, V18, N1.

[Knuth 86] Knuth D. E. (1986), *The TeXbook*, Computers & Typesetting A, Addison-Wesley, Reading, Massachusetts.

[Lamport 86] Lamport L. (1986), *LaTeX: A Document Preparation System*, Addison-Wesley, Reading, Massachusetts.

[Micro APL Ltd. 86] Micro APL Ltd. (1986), *APL.68000 for the Apple Macintosh*, Micro APL Ltd., London, England.

[The University of Chicago Press 82] University of Chicago Press (1982), *The Chicago Manual of Style* 13th *Edition*, The University of Chicago Press, Chicago, USA.

◇ Andreas Geyer-Schulz
Department of Applied Computer Science
Vienna University of Economics and Business Administration
Augasse 2–6
A-1090 Vienna, AUSTRIA
`ANDREAS@AWIWUW11.BITNET`

◇ Josef Matulka
Department of Applied Computer Science
Vienna University of Economics and Business Administration
Augasse 2–6
A-1090 Vienna, AUSTRIA
`MATULKA@AWIWUW11.BITNET`

◇ Gustaf Neumann
Department of Management Information Systems
Vienna University of Economics and Business Administration
Augasse 2–6
A-1090 Vienna, AUSTRIA
`NEUMANN@AWIWUW11.BITNET`

Experiments in TEXnicolour — A SLITEX Sub-style for Colour Printers*

David Love

Abstract

SLITEX assumes that colour transparencies will be produced from layers, each printed in a single colour. This sub-style allows a suitable `dvi` driver to produce multi-coloured slides in one go (i.e., using a single layer). It is customisable for PostScript printers or simpler colour devices like the the HP PaintJet.

1 Introduction

SLITEX—a version of LATEX for making slides—supports colours in its output by producing 'colour layers', one per colour [1]. It is assumed that these will be copied to transparencies of the appropriate colour and overlapped to make a composite multi-colour slide. Colour is valuable on slides, but this means of producing it is inconvenient and inappropriate if you have a colour printer and a suitable `dvi` driver for it.

This sub-style for SLITEX allows all the colours to be produced in one layer. It is designed for use with colour PostScript printers or others like the HP PaintJet with suitable `dvi` drivers, following experiments with these printers at Daresbury. It is intended to provoke discussion of the use of colour with TEX, a subject about which little has been said so far in public, rather than be at all definitive.

2 Approach

Use of colour with TEX is non-standard and has to be done by passing information to the `dvi` driver. This could be done in one of two ways:

- using fonts whose names idenitify them to the driver to be printed in a particular colour but are otherwise identical to the black version (possibly using virtual fonts [2]);
- using the `\special` mechanism to tell the driver to change colours.

The second option is easier to implement with an existing driver and allows colour for objects other than characters i.e., rules,[1] and has been adopted here. Unfortunately, existing drivers differ in their `\special` mechanisms and few understand colour explicitly. However, drivers which allow you to include arbitrary PostScript with a `\special` have this ability implicitly since colour operators are defined in PostScript [3]. (Colour gets rendered in shades of grey on a normal laser printer, which can help with proofing.)

There is a proposed standard for `\specials` [4] which includes colour information, but the definition of the `textcolor` `\special` is inconveniently related to the `dvi` stack. Working with existing PostScript drivers, one has to make some assumptions about them, as detailed below.

In the case of the PaintJet printer there does not appear to be an existing driver, so one was produced with a suitable `\special` defined arbitrarily.

3 PaintJet Driver

The Hewlett-Packard PaintJet can print in colour at 180 dpi, which is quite adequate for making transparencies. A driver for it was produced using the HP Laserjet version from Beebe's family [5] as a basis since this has rather similar control sequences. It was extended to maintain four bitmaps independently, allowing three colours other than black to be printed. Allowing more colours would make the code more complicated

*File version 1.2, dated 18/4/90.

[1] `\colorslides` doesn't work properly with rules—they come out in each layer.

and slower as well as using more memory for the bitmaps. It supports one `\special` to allow colour changes between black, red, green and blue. A defect is that the result of overlapping items of different colours is independent of their order in the `dvi` file, but this simplifies the driver considerably.

4 Assumptions

The necessary assumptions about the driver are largely isolated in the macros `\special@color` and `\colormix`.[2] Suitable changes could quite easily be made to conform with an eventual standard for `\special`s or use a different driver.

4.1 PostScript

We assume that a PostScript driver understands `\special`s of the form

`\special{ps::␣`⟨*code*⟩`}`,

where ⟨*code*⟩ is arbitrary PostScript from which the surrounding code output by the driver is not protected. This is appropriate for Rokicki's `dvips` and ArborText's DVILASER/PS. We use the RGB colour model [3, §4.8].

4.2 PaintJet or other printer

A non-PostScript printer should understand a special of the form `{color␣`⟨*colour*⟩`}`, where ⟨*colour*⟩ can take the values `black`, `red`, `green` or `blue`, determining the colour of all subsequent printing until the next change. The default is assumed to be black.

5 User interface

As well as redefining some internal SLiTeX macros, we make some new ones as described below.

`\psprinterfalse` The default is to assume the use of a PostScript printer. Saying `\psprinterfalse` changes this to produce output suitable for the PaintJet (or similar printer). This should be done only in the preamble.

`\colors`
`slide` Any colours you use *must* be declared using `\colors` (even red, green, blue and black). They do not need including in the argument of `\begin{slide}` for use with `\truecolors`, but there needs to be some argument, even if it's null.

`\truecolors` Like `\blackandwhite`, `\truecolors` is invoked from the driver file to process a batch of slides described in the file given as its argument. It works like `\blackandwhite` in that it makes only one layer, but will include the information for the printer to produce the colours in the `dvi` file. Notes are not printed by `\truecolors`.

5.1 PostScript specifics

`\colormix` Before you use a new PostScript colour, you have to define how it is mixed from red, green and blue. `\colormix` does this for you. It takes four arguments. The first is the name of the colour and the others are numeric values for the intensities of red, green and blue respectively in the mixture. (See the description of RGB in [3].) These values must lie in the range 0–1 such that 1 is most intense and 0 is no intensity. Thus you might define yellow by

`\colormix{yellow}{0}{1}{1}`

You don't need to define red, green, blue or black—they are done already.

You could put blocks of text on a coloured background either by setting them on top of a suitably-sized rule or by using explicit `\special`s.

[2] I spell the English way except in code names, where I reluctantly use 'color' for consistency with existing code.

5.2 Other printers

Only red, green, blue and black are available. Use of \colormix will produce a warning. Using a non-white background won't work very well with the PaintJet.

5.3 Problem with list environments

A problem occurs if you want the first text in a list environment item to be a different colour to the label since \specials before the text appear in the dvi file before the label. A way round this is to insert \leavevmode after \item. This sort of problem may occur in other situtaions.

6 Code

6.1 Driver-independent code

```
\typeout{SliTeX style option `truecols' (v. \fileversion\space of \filedate)}
```

\if@truecol We introduce a new switch to tell us whether we can assume we have true colour printing capability and unset it initially.

```
\newif \if@truecol \@truecolfalse
```

\the@color We keep track of the current colour using \the@color (to get colours right with grouping). Initially it's black.

```
\def\the@color{\black}
```

\ifpsprinter This tells us if we're dealing with a PostScript printer (the default) or not.

```
\newif \ifpsprinter \psprintertrue
```

\truecolors A new \truecolors command is defined, which works like \blackandwhite, printing everything in the file given as its argument in one layer. Thus it must set the \@bw switch. This slightly unfortunate change to \@bw's meaning makes the alterations easier.

```
\newcommand{\truecolors}[1]{\@truecoltrue \blackandwhite{#1}}
```

\@color We modify \@color to take note of the @truecol setting (if @bw is set). If appropriate, it puts out a colour-changing \special using \special@color. We can omit the test for a colour change in math mode if we're not depending on invisible fonts.

```
\def\@color#1{%
  \if@truecol \else \@mmodetest \fi %  change
  \if@bw \@visibletrue
    \if@truecol \special@color{#1}\fi %  change
  \else \@visiblefalse
    \@for \@x@a :=#1\do{\ifx\@x@a\@currcolor\@visibletrue\fi}\fi
  \@currsize \@currfont \ignorespaces}
```

\endslide We ensure that the driver colour is reset to black at the end of each slide by setting it in \endslide.

```
\def\endslide{\@color{black}\par\break}
```

\note We don't want to waste truecolour output on producing notes, which we will if we don't re-define \note (since @bw is set by \truecolors). We just add a test on @truecol.

```
\def\note{%
  \stepcounter{note}%
  \if@bw
    \if@truecol \gdef\@slidesw{F}%  added
    \else
      \gdef\@slidesw{T}%
      \if@onlynotesw\@whilenum \c@slide > \@donotehigh\relax
        \do{\@setlimits\@donotelist\@donotelow\@donotehigh}\ifnum
      \c@slide < \@donotelow\relax \gdef\@slidesw{F}\fi\fi\fi
  \else \gdef\@slidesw{F}\fi
  \if\@slidesw T\newpage\thispagestyle{note}\else
  \end{note}\@gobbletoend{note}\fi }
```

`\colors` We re-define `\colors` to check that we know about those specified. It could, perhaps, be dispensed with at the cost of incompatibility with standard SLiTeX. If we're not using PostScript only the four pre-defined colours are available.

```
\def\colors#1{%
  \@for\@colortemp:=#1\do{%
    \@ifundefined{\@colortemp @mix}{% change
      \ifpsprinter
        \errhelp{You could use I to define it now.}%
        \errmessage{You need to use \noexpand\colormix to define
                    \@colortemp}%
      \else
        \errhelp{It's safe to carry on.}%
        \errmessage{You can only use black, red, green and blue with
                    this printer, not \@colortemp}%
        \expandafter\xdef\csname\@colortemp\endcsname{\relax}%
      \fi
    }{}%
    \expandafter\xdef\csname\@colortemp\endcsname
      {\noexpand\@color{\@colortemp}}%
  }% (do)
  \ifx\@colorlist\@empty \gdef\@colorlist{#1}%
  \else \xdef\@colorlist{\@colorlist,#1}\fi}
```

`\pagestyle` Finally we change the default pagestyle since the alignment marks would only be useful with overlays.

```
\pagestyle{plain}
```

6.2 Driver-dependent code

`\colormix` The PostScript user can mix new colours in the RGB model using `\colormix`. It takes four arguments, the name of the colour (which must then be declared with `\colors`) and three real numbers in the range 0–1 describing respectively the intensity of red, green and blue in the result. The result is to define a macro of the form `\`⟨*colour*⟩`@mix` to be a list of arguments 2–4.

```
\def\colormix#1#2#3#4{%
  \ifpsprinter \else \typeout{Warning: \noexpand\colormix used with
    \noexpand\psprinterfalse.}\fi
  \color@range@check{#2}\color@range@check{#3}\color@range@check{#4}%
  \expandafter\xdef\csname#1@mix\endcsname{#2 #3 #4}}
```

`\color@range@check` We need to check that the numeric arguments of `\colormix` are in the range 0–1. Since they're real numbers we convert them to dimensions to test.

```
\def\color@range@check#1{%
  \def\fred{\errhelp{Carry on, but you'll get a PostScript error if
              you try to print the results.}%
            \errmessage{\noexpand\colormix arguments must be in the
              range 0--1}}%
  \ifdim#1 pt < 0pt \fred \fi \ifdim#1 pt > 1pt \fred \fi }
```

`\special@color` This has to change the current colour on the output device to that given as its argument, but first has to store the current value and make sure it is reinstated at the end of the current group. (Coloured fonts would avoid the need for this.) Note the

`{}` inserted at the end of the group to avoid spaces being gobbled afterwards as with `{\red foo} bar`. Probably there's a better way of doing this.

```
\def\special@color#1{%
  \def\the@color{\csname#1\endcsname}\aftergroup\the@color
    \aftergroup{\aftergroup}%
```

To change colour with PostScript we use a `\special` which will emit code of the form

⟨*red level*⟩␣⟨*green level*⟩␣⟨*blue level*⟩␣`setrgbcolor`

Perhaps this should be parameterised. The three levels are provided by a macro of the form `\`⟨*colour*⟩`@mix` which must be defined for each ⟨*colour*⟩ which is used. It might be useful to define a white font (RGB=[1 1 1]) which could be painted over a coloured region.

```
    \ifpsprinter \special{ps:: \csname#1@mix\endcsname\space
                                setrgbcolor\space}%
```

For the PaintJet we output

`\special{color`␣⟨*colour*⟩`}`,

where ⟨*colour*⟩ can be `black`, `red`, `green` or `blue` and we assume that `#1` is one of these.

```
    \else \special{color #1}\fi % (\ifpsprinter)
  } % (\def\special@color)
```

`\black@mix` `\red@mix` `\blue@mix` `\green@mix` Here we define the RGB mixes for the PostScript colours we support initially. These macros need defining for the PaintJet too, so that we can check on colours the user tries to invoke.

```
  \def\black@mix{0 0 0} \def\red@mix{1 0 0}
  \def\blue@mix{0 0 1}  \def\green@mix{0 1 0}
```

References

[1] Leslie Lamport. *LaTeX: A Document Preparation System*; `slitex.tex` dated 10 November 1986; `slides.sty` dated 17 January 1986.

[2] Donald Knuth. "Virtual fonts: More fun for grand wizards," *TUGboat*, 11(1), 1990, p. 13.

[3] Adobe Systems Inc. *PostScript Language Reference Manual.* Addison-Wesley, 1985, ISBN 0-201-10174-2.

[4] Don Hosek. Proposed DVI `\special` command standard. (See *TUGboat*, 10(2), 1989, p. 188.)

[5] Nelson H. F. Beebe. "Public Domain TeX DVI Driver Family," *TUGboat* 8(1), p. 41.

◇ David Love
SERC Daresbury Laboratory,
Warrington WA4 4AD, UK
`d.love@daresbury.ac.uk`

Footnotes in a Multi-Column Layout*

Frank Mittelbach

1 Introduction

The placement of footnotes in a multi-column layout always bothered me. The approach taken by LaTeX (i.e., placing the footnotes separately under each column) might be all right if nearly no footnotes are present. But it looks clumsy when both columns contain footnotes, especially when they occupy different amounts of space.

In the multi-column style option [5], I used page-wide footnotes at the bottom of the page, but again the result doesn't look very pleasant since short footnotes produce undesirable gaps of white space. Of course, the main goal of this style option was a balancing algorithm for columns which would allow switching between different numbers of columns on the same page. With this feature, the natural place for footnotes seems to be the bottom of the page,[1] but looking at some of the results it seems best to avoid footnotes in such a layout entirely.

Another possibility is to turn footnotes into endnotes, i.e., printing them at the end of every chapter or the end of the entire document. But I assume everyone who has ever read a book using such a layout will agree with me that it is a pain to search back and forth, so that the reader is tempted to ignore the endnotes entirely.

When I wrote the article about future extensions of TeX [6], I was again dissatisfied with the outcome of the footnotes, and since that article was to show certain aspects of high quality typesetting, I decided to give the footnote problem a try and modified the LaTeX output routine for this purpose. The layout I used was inspired by the yearbook of the Gutenberg Gesellschaft Mainz [1]. Later on, I found that it is also recommended by Jan White [9]. On the layout of footnotes I also consulted books by Jan Tschichbold [8] and Manfred Simoneit [7], books I would recommend to everyone able to read German texts.

1.1 Description of the new layout

The result of this effort is presented in this paper and the reader can judge for himself whether it was successful or not.[2] The main idea for this layout is to assemble the footnotes of all columns on a page and place them all together at the bottom of the right column. Allowing for enough space between footnotes and text, and, in addition, setting the footnotes in smaller type,[3] I decided that one could omit the footnote separator rule which is used in most publications prepared with TeX.[4] Furthermore, I decided to place the footnote markers[5] at the baseline instead of raising them as superscripts.[6]

All in all, I think this generates a neat layout, and surprisingly enough, the necessary changes to the LaTeX output routine are nevertheless astonishingly simple.

1.2 The use of the style option

This style option might be used together with any other style option for LaTeX which does not change the three internals changed by `ftnright.sty`.[7] In most cases, it is best to use this style option as the very last option in the `\documentstyle` command to make sure that its settings are not overwritten by other options.[8]

It is unfortunate that the current LaTeX has no provisions to make such changes without overwriting the internal routines. In the new LaTeX implementation, we will certainly add some hooks that will make such changes more easy.

The `ftnright` option makes use of the values of `\textheight` and `\skip\footins` (the space between text and footnotes). The values used are the ones current when `ftnright.sty` is read in. If the user wants to change either of them in the preamble of his document he should call the macro

*. The LaTeX style option `ftnright` which is described in this article has the version number v1.0c dated 90/08/24. The documentation was last revised on 90/10/01.

1. You cannot use column footnotes at the bottom, since the number of columns can differ on one page.

2. Please note that this option only changed the placement of footnotes. Since this article also makes use of the `doc` option [4] that assigns tiny numbers to code lines sprinkled throughout the text, the resulting design is not perfect.

3. The standard layout in *TUGboat* uses the same size for footnotes and text, giving the footnotes, in my opinion, much too much prominence.

4. People who prefer the rule can add it by redefining the command `\footnoterule` [2, p. 156]. Please note that this command should occupy no space, so that a negative space should be used to compensate for the width of the rule used.

5. The tiny numbers or symbols appearing with the footnote text; e.g., the '5' in front of this footnote.

6. Of course, this is only done for the mark preceeding the footnote text and not the one used within the main text where a raised number or symbol set in smaller type will help to keep the flow of thoughts uninterrupted.

7. These are the macros `\@startcolumn`, `\@makecol` and `\@outputdblcol` as we will see below. Of course, the option will only take effect with a document style using a twocolumn layout (like `ltugboat`) or when the user additionally specifies `twocolumn` as a document style option in the `\documentstyle` command.

8. The `ltugboat` option (which is currently set up as a style option instead of a document style option which it actually is) will overwrite the size used in footnotes if it follows the `ftnright` option.

`\preparefootins` afterwards to reinitialize the footnote algorithm, e.g.,

```
\setlength{\skip\footins}{8pt plus 3pt}
\addtolength{\textheight}{1in}
\preparefootins
```

This is necessary because the current LaTeX version contains no hook at the `\begin{document}` command where we could force an execution of `\preparefootins` internally.

2 The Implementation

As usual, we start by identifying the current version of this style file in the transcript file.[9]

```
\wlog{Style Option: `\filename'
  \fileversion\space <\filedate> (FMi)}
\wlog{English Documentation
  \@spaces\@spaces\space <\docdate> (FMi)}
```

To implement the layout described above, we have to distinguish between the left and the right column on a page. For this purpose LaTeX maintains the switch `\if@firstcolumn`. When assembling material for the left (i.e., the first) column, footnotes should take up no space, since they are held over for the second column. In the second column these footnotes are combined with the ones found there and placed a suitable distance from the main text at the bottom of this column.

This means that we have to change certain parameters for the insertion `\footins` when we construct the second column. The right place to do this is in the LaTeX macro `\@outputdblcol` which we are going to change later on. What settings for the insertion parameters are appropriate? For setting the first column, `\count\footins` and `\skip\footins` should both be zero since footnotes are held over, while for the second column `\count\footins` should be[10] 1000 and the `\skip\footins` has to be set to the desired separation between main text and footnotes.

We will allow one column of footnotes (i.e., the right column) at most, so that `\dimen\footins` has to equal `\textheight`. In principle, it would be possible to allow for even more footnotes, but this would complicate matters enormously.[11]

Since a document usually starts with a left column, we have to set `\count` and `\skip\footins` on top-level to zero. For this purpose, we define a macro `\preparefootins` which will first save the current value of `\skip\footins` in a safe place. This saved value will be used later for the second column. In this way, it is possible for the user or a designer of a document style to adjust this parameter without fiddling with the code of this style file.

```
\def\preparefootins{%
  \global\rcol@footinsskip\skip\footins
  \global\skip\footins\z@
  \global\count\footins\z@
```

We will also assign `\textheight` to `\dimen \footins` to allow the user to change this parameter in the preamble.

```
  \global\dimen\footins\textheight}
```

It is necessary to make the assignments above `\global` because we are going to use this macro in the output routine which has an implicit grouping level to keep the changes made by it local.

Of course, we have to allocate the skip register that we used above:

```
\newskip\rcol@footinsskip
```

Now we have all the necessary tools available to tackle `\@outputdblcol`. We have to remember that when `\if@firstcolumn` equals `\iftrue`, we are currently starting to build the second column, i.e., that the first column is already assembled. Therefore, the macro will start with the following code:

```
\def\@outputdblcol{\if@firstcolumn
  \global\@firstcolumnfalse
```

After changing the switch, we save the first column (which was placed by preceeding macros in `\@outputbox`) in the box register `\@leftcolumn`. Since we are inside the output routine, all those assignments have to be `\global` to take any effect.

```
  \global\setbox\@leftcolumn\box\@outputbox
```

9. Nico Poppelier suggested omitting the `\typeout` statements in the production version of the files to avoid showing all that unnecessary information to the user. While I accept his criticism as valid, I decided that this information should at least be placed into the transcript file to make it easier to detect problems arising from the use of older versions. The command `\wlog` is a PLAIN TeX command that will write its argument to the transcript file.

10. A value of 1000 means that there is a one-to-one relationship between the real size of the footnote and the size finally occupied by the footnote on the current page.

11. If one likes to allow that the footnote text might start in the first column occupying also the whole second column, it is not possible to simply make `\dimen\footins` larger than `\textheight` directly, because this would result in a full left column (with text) and more than one column of footnotes which will not fit on the current page. Instead, one has to make footnotes visible to the page generation algorithm again at the moment when a full column of footnotes is assembled, but we still have some space left in the first column. It is a nice enhancement, and, I suppose, it is of some value for preparing publications in certain disciplines, so here is the challenge ...

exact. If we have a full column of footnotes, it will be too high, but this does not matter since we need it only for an upper bound on the free space available for floats.

```
      \ftn@amount\ht\footins
      \advance\ftn@amount\dp\footins
      \advance\ftn@amount\skip\footins
    \fi
```

We then reduce the `\@colht` by this amount and again assign `\@colroom` the value of `\@colht`. If no footnotes are present, we substract zero, so there is no harm in doing this operation all the time.

```
    \global\advance\@colht-\ftn@amount
    \global\@colroom\@colht
```

Now, we call another internal LaTeX macro that will try to contribute floats to the next column. It will use the register `\@colht` when trying to build up a float column, which is the reason for reducing this register. If it succeeds, it will set the switch `\if@fcolmade` to `true`, otherwise, to `false`. If no float column is possible, it will try to place some or all of the deferred floats to the top or the bottom of the next column, thereby, using and reducing the value of the register `\@colroom`.

```
    \@xstartcol
```

Afterwards, we have to restore the correct values for `\@colht` and `\@colroom` again, but this time, they may differ, so that we have to `\advance` both registers separately by `\ftn@amount`.

```
    \global\advance\@colht\ftn@amount
    \global\advance\@colroom\ftn@amount
  \fi
```

Now, after doing the things depending on the status of the `\@deferlist`, we have to incorporate the left over footnotes in the new column. First we check whether a float column was produced by `\@xstartcol` or not.

```
  \if@fcolmade
```

If so, we do something awful. To make use of the `\@makecol` macro, which attaches footnotes to `\box255` and places the result in the box register `\@outputbox`, we have to assign `\@outputbox` (i.e., the result of `\@xstartcol`) to `\box255`.[13]

```
    \setbox\@cclv\box\@outputbox
    \@makecol
  \else
```

If no float column was produced, we reinsert the held over footnotes so that they can be reconsidered by the page generation algorithm of TeX. But it is necessary to ensure that this operation is done only when footnotes are actually present.[14]

```
     \ifvoid\footins\else
       \insert\footins{\unvbox\footins}\fi
  \fi}
```

Of course, we also have to allocate the dimen register. It will be automatically initialized to zero.

```
\newdimen\ftn@amount
```

The other internal macro that we have to change is `\@makecol`, a macro that is called whenever one column of material is assembled and column floats and footnotes have to be added. Again, we have to distinguish between actions for the first and the second column.

```
\def\@makecol{\if@firstcolumn
```

For the first column, we leave the footnotes in their box and simply save the contents of `\box255` in the `\box` register `\@outputbox`.

```
  \setbox\@outputbox\box\@cclv
```

But if the user erroneously forgot to specify a twocolumn layout, we will always typeset the first column, so that the footnotes are never printed. Therefore we better check for this special case and output the footnotes on a separate page in an emergency.[15]

```
  \if@twocolumn \else
    \ifvoid\footins \else
      \@latexerr
   {ftnright option used in one-column mode}%
   {I shipped out the the footnotes on an
    extra page.}%
      \shipout\box\footins \fi\fi
 \else
```

When we construct the second column, we must first check whether footnotes are actually present. If not, we perform the same actions as before.

```
  \ifvoid\footins
    \setbox\@outputbox\box\@cclv
  \else
```

But, if footnotes are present, it may be possible that the whole column consists of footnotes, i.e., `\box255` is empty. In this case, there is no use in placing

13. In German, we call this "from the back through the chest into the eyes".

14. Otherwise, we might get an undesired extra vertical space coming from `\skip\footins`, even if there are no footnotes on the page.

15. Otherwise, the footnotes are held over forever, preventing TeX from finishing the document successfully. Instead, TeX *will produce infinitely many empty pages at the end of* the document, trying in vain to output the held over footnotes. This problem was found by Rainer Schöpf when we prepared the paper for the Cork conference.

Then, we make the footnotes visible to the page generation algorithm by setting `\count\footins` to 1000 (`\@m` is an abbreviation for this number) and `\skip\footins` to its saved value (i.e., `\rcol@footinsskip`).

```
  \global\count\footins\@m
  \global\skip\footins\rcol@footinsskip
```

We also have to reinsert all footnotes left over from the first column to make sure that they are reconsidered by the page generation algorithm of TEX using the new values for `\count` and `\skip\footins`. But this will be done later in the macro `\@startcolumn`.

If we have just finished the right column, i.e., when `\if@firstcolumn` equals `\iffalse`, we will reset the `\footins` parameters as explained above using the utility macro `\preparefootins`.

```
\else \preparefootins
```

Then, we compose both columns in `\@outputbox`, combine them with all page-wide floats for this page (`\@combinedblfloats`), attach header and footer, and ship out the result (`\@outputpage`). Finally we look to see whether it is possible to generate following pages consisting only of page-wide floats.[12]

```
 \global\@firstcolumntrue
 \setbox\@outputbox\vbox{\hbox to\textwidth
   {\hbox to\columnwidth
                 {\box\@leftcolumn\hss}%
    \hfil\vrule\@width\columnseprule\hfil
    \hbox to\columnwidth
                 {\box\@outputbox\hss}}}%
 \@combinedblfloats\@outputpage
 \begingroup
  \@dblfloatplacement\@startdblcolumn
  \@whilesw\if@fcolmade\fi
    {\@outputpage\@startdblcolumn}%
 \endgroup
 \fi}
```

There is a fundamental flaw in LATEX's output routine for float columns and float pages: split footnotes, i.e., footnotes which are only partly typeset on the preceding page are not resolved. They are held over until LATEX starts a page (or column) containing text besides floats again. For our current layout, this would mean, that if LATEX decided to make the right column of a page a float column, footnotes from the left column would appear on a later page. A real cure for this problem would be to rewrite two-thirds of LATEX's output routine, so I am leaving this open for the interested reader.

But the problem shows up even if only one float is contributed to the right column since LATEX assumes that the whole column is usable, whereas some of it might actually be already devoted to footnotes from the left column. So we have to change the output routine at least in the part that contributes floats to the next column. The macro involved is called `\@startcolumn`. The first thing we do is to check whether any deferred floats exists.

```
\def\@startcolumn{%
\ifx\@deferlist\@empty
```

If not, we set the switch `\if@fcolmade` to `false` which says that we did not succeed in making a float column. Then, we set `\@colroom` to `\@colht`. The register `\@colht` holds the amount of space that is available for floats, text, and footnotes in one column, i.e., it equals `\textheight` minus the space devoted to page-wide floats. `\@colroom` is a similar register which holds the value `\@colht` minus space for column floats that are already contributed to the current column. Of course, both values should be equal when we start a new column.

```
  \global\@fcolmadefalse
  \global\@colroom\@colht
\else
```

If there are floats waiting for a change to be processed, the situation is more difficult. In this case, we have to reduce both `\@colht` and `\@colroom` by the amount of space that will be needed for the footnotes from the left column. So we must check whether such footnotes are present. As we have not reinserted them in `\@outputdblcol`, we can check the `\footins` box.

```
  \ifvoid\footins\else
```

If there are some, we measure the space that will be occupied by them. This measurement is not really

12. This part is copied directly from the original LATEX macro. Details about the macros used, their interfaces and meanings can be found in the LATEX source code [3].

Puzzle:

Given a simple TEX document containing only straight text, is it possible for the editor, after deleting one sentence, to end up with a document producing an extra page?

We assume that the deleted text contains no TEX macros and that the document was prepared with a standard macro package like the one used for *TUGboat* production.

The answer will be given in the next issue.

any glue (`\skip\footins`) in front,[16] so we have to check for this possibility.

```
  \setbox\@outputbox\vbox
    {\ifvoid\@cclv \else
       \unvbox\@cclv
       \vskip\skip\footins\fi
```

But in any case, we place the `\footnoterule` in front of the footnotes even if this macro is not used by this style option.[17] This ends the if-statement testing whether footnotes are present or not. It also ends the code which differs depending on the column number.

```
     \footnoterule\unvbox\footins}\fi
  \fi
```

Now the column floats are added at the top and the bottom, and the `\@outputbox` is adjusted to the full column height so that the glue inside will stretch in certain situations.[18] Again, this code is copied verbatim from the original source, so I won't dwell on details.[19]

```
  \xdef\@freelist{\@freelist\@midlist}%
  \gdef\@midlist{}\@combinefloats
  \setbox\@outputbox\vbox to\@colht
     {\boxmaxdepth\maxdepth
      \@texttop
      \@tempdima\dp\@outputbox
      \unvbox\@outputbox
      \vskip-\@tempdima
      \@textbottom}%
  \global\maxdepth\@maxdepth}
```

Now we can tackle the remaining small changes to the standard layout. I decided to use a smaller size for footnotes but with a slightly larger leading than usual.[20] This means that we have to redefine the `\footnotesize` macro which depends on options like `11pt`, etc. Fortunately, there is a simple way to find out the main size of the document: the macro `\@ptsize` contains 0, 1, or 2 standing for 10, 11, or 12 points document text size.

```
\ifcase \@ptsize
\def\footnotesize{\@setsize\footnotesize
   {9.9pt}\viiipt\@viiipt}
\or
\def\footnotesize{\@setsize\footnotesize
   {11.1pt}\ixpt\@vixpt}
\or
\def\footnotesize{\@setsize\footnotesize
   {12.3pt}\xpt\@xpt}
\fi
```

Setting footnotes in smaller type and separating them with sufficient space from the main text allow us to omit the `\footnoterule` normally used.

```
\let\footnoterule\@empty
```

Individual footnotes are separated from each other by approximately a `\baselineskip` of the text size. This can be specified with the following code:

```
{\normalsize
\global\footnotesep\ht\strutbox}
```

Braces and `\global` were used to keep the switch to `\normalsize` local, just in case some weird layout starts out with a different text size for some reason.

And finally, a small but nice change, to the mark at the beginning of the footnote text. We will place it at the baseline instead of raising it as a superscript. Additionally, it will get a dot as punctuation.

```
\long\def\@makefntext#1{\parindent 1em
  \noindent\hbox to 2em{}%
  \llap{$\@thefnmark.\;\;$}#1}
```

3 Initialisation

We defined the macro `\preparefootins` above, but we also have to use it to prepare typesetting the first column. As a default for the separation of footnotes and text on the second column, we use the following:

```
\skip\footins 10pt plus 5pt minus 3pt
\preparefootins
```

Of course, this value can be changed later on by the user as described in the introduction.

16. In fact, it would be a mistake since this glue was not taken into account when the footnotes where assembled, so it would produce an overfull box.

17. This decision is certainly open to criticism, since there is nothing to separate. On the other hand, a rule or some other ornament in front of the footnotes is part of the design which should be used consistently throughout a document. As a last argument in favor of the rule, consider the situation where LaTeX decided to place only floats and footnotes into the right hand column. In this case a separator again seems adequate. In this situation one can even argue that it is necessary to put in the `\skip\footins`.

18. It is an interesting question as to whether the current layout works well with bottom floats or not. Actually, I would prefer to place the footnotes below the bottom floats instead of above, as it is done here. At least when the floats are part of the document and not puzzles thrown in. But I was too lazy to implement it because I seldom use floats. If somebody implements this layout (some parts of this macro have to be changed) I would be interested in seeing the code and some sample results.

19. I only changed `\dimen128` into `\@tempdima` which is, besides being faster and shorter, only a cosmetic change. The use of this hardwired dimen register seems to indicate that this part of LaTeX was written very early and left unchanged since then: an interesting fact for software archeologists.

20. The sizes used are suitable for high resolution printers but should be perhaps enlarged for resolutions of 300dpi or less. On such output devices footnotes of 8pt size are difficult to read and look too small.

References

[1] Hans-Joachim Koppitz, editor. *Gutenberg Jahrbuch.* Gutenberg-Gesellschaft, Mainz.

[2] Leslie Lamport. *LaTeX: A Document Preparation System.* Addison-Wesley, Reading, Massachusetts, 1986.

[3] Leslie Lamport. *latex.tex*, February 1990. LaTeX source version 2.09.

[4] Frank Mittelbach. The doc-option. *TUGboat*, 10(2):245–273, July 1989.

[5] Frank Mittelbach. An environment for multicolumn output. *TUGboat*, 10(3):407–415, November 1989.

[6] Frank Mittelbach. E-TeX: Guidelines to future TeX extensions. In Lincoln K. Durst, editor, *TUGboat*, 11(3): *1990 TUG Annual Meeting Proceedings*, pages 337–345, September 1990.

[7] Manfred Siemoneit. *Typographisches Gestalten.* Polygraph Verlag, Frankfurt am Main, second edition, 1989.

[8] Jan Tschichbold. *Ausgewählte Aufsätze über Fragen der Gestalt des Buches.* Birkhäuser Verlag, Basel, 1987. Second printing.

[9] Jan White. *Graphic Design for the Electronic Age.* Watson Guptill, Xerox Press, New York, 1988.

4 Index

All numbers denote code lines where the corresponding entry is used, underlined entries point to the definition.

◇ Frank Mittelbach
Electronic Data Systems
(Deutschland) GmbH
Eisenstraße 56 N15
D-6090 Rüsselsheim
Bitnet: pzf5hz@drueds2

Abstracts

Deutsche Kurzfassungen der *TUGboat*-Artikel [German Abstracts of *TUGboat* Articles]

Luzia Dietsche

Makros zum Schreiben von Kreuzworträtseln (B. Hamilton Kelly, *TUGboat* 11(1), S. 103)

Immer öfter erscheinen in Computer Zeitschriften Kreuzworträtsel zur Unterhaltung der Leser. Für solche Fälle hat der Autor die `crossword` Umgebung geschaffen, die zusammen mit LaTeX verwendbar ist. Die Makros bieten die Möglichkeit, sowohl leere Kreuzworträtsel als auch deren Lösung zu setzen. Als zusätzliches Bonbon wird überprüft, ob das Gitter des Kreuzworträtsels richtig zusammenpaßt.

Die Zukunft von TeX (S. 488)

In Texas und Cork fanden birds-of-a-feather Sitzungen statt, die die Zukunft von TeX zum Thema hatten. Daraus resultierten vier Fragen die Betreuung und Weiterentwicklung von TeX betreffend, die an das TUG Board gerichtet waren. Auch Prof. D. Knuth bekam die Fragen zugeschickt. Unterzeichnet ist das Papier von 25 Personen aus 7 Ländern.

Die Zukunft von TeX und METAFONT (D. Knuth, S. 489)

D. Knuth bezieht Stellung zu den vier Fragen betreff der Zukunft von TeX. Er stellt fest, daß seine Arbeit an TeX, METAFONT und Computer Modern zu einem Ende gekommen ist. Er wird weiterhin Fehler in TeX 3.1 und METAFONT 2.7 korrigieren und die neuen Versionen über den Hauptserver für TeX zur Verfügung stellen. Ab dem Zeitpunkt seines Todes sollen TeX und METAFONT für immer unverändert bleiben. Jeder Person ist es freigestellt, entsprechend den Copyright-Vermerken in Volume B, D und E die zugrunde liegenden Programme zu verwenden, solange für das Ergebnis nicht der Name TeX, METAFONT oder Computer Modern verwendet wird. D. Knuth wird die Verantwortung für diese drei Pakete niemals an jemand anderen abgeben. Die Weiter- oder Neuentwicklung von Systemen, die besser zum Setzen von Texten geeignet sind, wird von ihm begrüßt, vorausgesetzt sie bedienen sich nicht der durch das Copyright geschützten Namen.

Kommentare zur Zukunft von TeX und METAFONT (N. H. F. Beebe, S. 490)

Nelson Beebe zeigt die Stellung der TUG nach der Veröffentlichung von D. Knuth über die Zukunft von TeX und METAFONT auf, beschreibt die Internationalität von TeX, die Herausforderung durch desktop publishing Systeme, die Antwort von TeX darauf und einige Beobachtungen in Bezug auf kommerzielle Anbieter. Daraus leitet er notwendige zukünftige Entwicklungen ab, die durch die Bestimmungen von D. Knuth die Namen TeX, METAFONT und Computer Modern nicht verwenden dürfen und eventuelle Inkompatibilitäten in Kauf nehmen.

Die Antworten zu den Übungen zu *TeX: The Program* (D. Knuth, S. 499)

Nachdem in *TUGboat* 11, no. 2, pp. 165–170 verschiedene Übungsaufgaben vorgestellt wurden, die der „Grand Wizard of TeX" seinen Studenten zu *TeX: The Program* gestellt hatte, werden nun die Lösungen dazu offenbart. Allerdings sollte man die Aufgabenstellungen zur Hand haben, da tatsächlich nur die Lösungen abgedruckt sind. Und die sind ohne Fragen teilweise kryptisch.

Literarisches Programmieren ohne `WEB` (J. Fox, S. 511)

In diesem Artikel wird c-~~web~~ (auch *no-web*) als Alternative zum System *CWEB — Literarisches Programmieren* eingeführt. c-~~web~~ ermöglicht es einem Software-Entwickler, für ein und denselben Quelltext sowohl `tex` (formatieren) als auch `cc` (compilieren) aufzurufen, ohne einen zusätzlichen Präprozessor zu benötigen.

Eine Umgebung, um METAFONT in PostScript zu übersetzen (S. Yanai & D. M. Berry, S. 525)

Hier wird ein Programm beschrieben (`mf2ps`), das eine METAFONT Schriftdefinition in eine Definition für dieselbe Schrift in PostScript-Sprache übersetzt. `mf2ps` wurde aus dem Teil des METAFONT Programms konstruiert, das die „Hüllen" der Buchstaben herauslöst; diese „Hüllen" werden in PostScript-Umrisse übertragen.

Eplain (K. Berry, S. 571)

Eplain ist ein Makropaket, basierend auf `plain`, das vom Autor für das Buch *TeX for the Impatient* entwickelt wurde. Der Name steht für *extended plain.* Das Paket ist frei und wird von einer 20-Seiten starken Dokumentation begleitet.

Die IVRITEX Diskussions-Liste (D. Hosek, S. 578)

Zusätzlich zu den bereits vorhandenen Listen für alles, was TEX angeht, wurde IVRITEX eingerichtet. Diese Liste ist vor allem für Benutzer gedacht, die TEX mit Hebräisch, Arabisch oder ähnlichen Sprachen verwenden. Zweiwöchig wird eine Zusammenfassung der Entwicklungen an die Liste geschickt. Die Liste ist bei `listserv@taunivm.bitnet` eingerichtet.

Output Routinen: Beispiele und Techniken. Teil III: „Einfügungen" (D. Salomon, S. 588)

Der `\insert`-Mechanismus wird von vielen Anwendern, die sehr wohl mit „token", Makros oder Output Routinen umgehen, tunlichst umgangen. Der Grund dafür liegt darin, daß im TEXbuch kaum Erklärungen, geschweige denn Beispiele für Einfügungen wie z.B. Fußnoten gegeben sind. Daher ist diese Folge der Artikelreihe der detaillierten Beschreibung des `\insert`-Befehls in Verbindung mit speziellen und einfachen Beispielen gewidmet.

Auch Teil III der Serie sollte nicht ohne die beiden vorhergehenden gelesen und angewendet werden.

Ein Schema für die Verwendung von `\parindent` (V. Eijkhout, S. 613)

In diesem zweiten Artikel beschreibt Victor Eijkhout den Mechanismus von Absatzeinrückungen. Zugegebenermaßen einfacher als die Änderung von Paragraphabständen, sollten auch Einrückungen nicht durch bloßes Setzen auf Null beeinflußt werden. Erneut wird dem Leser eine verfeinerte Handhabung des Problems dargestellt. Die vorgeschlagene Lösung erlaubt die Verwendung von eingerückten oder nicht-eingerückten Paragraphanfängen ohne den Befehl `\noindent`.

Ein Schema für die Veränderung von `\parskip` (V. Eijkhout, S. 616)

Viele Anwender von TEX/LATEX bevorzugen ein Layout, das den Wert für Absatzeinrückungen auf Null und dafür den Wert für den Paragraphenabstand höher setzt. Dieses Ziel verwirklichen die meisten durch zwei Zeilen, nämlich die Umsetzung der beiden Werte in der Präambel. Das hat aber unerwartete Nebenwirkungen auf viele andere Stellen. Deshalb beschreibt der Autor eine Methode, die für ein ganzes Schriftstück Gültigkeit hat.

Arbeiten mit `\afterassignment` (S. Maus, S. 612)

Das Primitiv `\afterassignment` kann u.a. dazu benutzt werden, um einer Variablen einen Wert zuzuweisen und diese Variable danach zu benutzen. Das hat den Vorteil, daß TEX bei der Ausführung eines Befehls, in dem die Variable vorkommt, schon den zugewiesenen Wert kennt. Besonders im Zusammenhang mit Boxen erweist sich dieses Prinzip als nützlich.

Zeilenumbruch in `\unhboxed` Text (M. Downes), S. 605

Michael Downes sties im Zusammenhang mit dem Mechanismus im Zeilenumbruch auf einen Fehler in den Bibliographie-Makros von `amsppt.sty`, der allem Anschein nach bisher niemandem aufgefallen war. Hier nun beschreibt er das Problem und die mögliche Lösung, die er zusammen mit B. Beeton und D. Knuth gefunden hat.

Wie sichert man Kommandos bei der Verwendung von `\write` (R. Whitney, S. 620)

Dieser Artikel ist, genauso wie der nächste, eher für die Entwickler von Stylefiles, denn deren Anwender gedacht. Der `\write`-Mechanismus stellt die Möglichkeit zur Verfügung, Informationen für eine spätere Verarbeitung in ein anderes File zu schreiben. Dabei werden Kommandos, die übergeben werden, sofort ausgeführt. Wie man das verhindert, beschreibt Ron Whitney in sehr kompakter Form.

Versuche in TEXnicolor — Ein Stylefile zu SliTEX für Farbdrucker (D. Love, S. 652)

SLITEX geht davon aus, daß farbige Folien aus mehreren Schichten (von Folien) bestehen, wobei jede einzelne Schicht in einer eigenen Farbe gedruckt wird. Der in dem Artikel vorgestellte Stylefile erlaubt es, zusammen mit einem passenden `.dvi`-Treiber, mehrfarbige Folien in einem einzigen Arbeitsgang zu produzieren. Anwendbar ist diese Methode bei PostScript-Druckern, aber auch bei einfacheren Ausgabegeräten wie z.B. dem HP Paint Jet.

Eine Style-Option, um *APL* zu setzen (A. Geyer-Schulz, et al., S. 644)

Die Autoren beschreiben die Style-Option `apl.sty`, mit der Text zusammen mit *APL*-Code gesetzt werden kann. Alle Symbole, die in solch einer Umgebung benötigt werden, sind vorhanden. Sie sind der Standard-Schriftfamilie von LaTeX entnommen. Dadurch werden keine zusätzlichen Fonts nötig. Mit normalen LaTeX-Kommandos kann die Schriftgröße solcher Symbole beeinflußt werden. Durch einen in *APL* geschriebenen Präprozessor können *APL* Objekte automatisch umgewandelt werden.

Überblick über EDMAC: Ein plain Format für kritische Textausgaben (J. Lavagnino & D. Wujastyk, S. 623)

EDMAC besteht aus einer Sammlung von plain Makros, die es ermöglichen, kritische Textausgaben traditioneller Art wie es z.B. bei Oxford Classical Texts, Teubner, Arden Shakespeare oder anderen Reihen üblich ist, zu formatieren. Aufgenommen wurde an Grundfunktionen die Möglichkeit der Numerierung von Zeilen am Rand und mehrere Zählungen von Fuß- und Endnoten bezogen auf die Zeilennummern. Da der interne Arbeitsablauf von EDMAC gezwungenermaßen esoterisch ist, sind die dazugehörenden Makros relativ einfach gehalten. Dadurch kann jeder die genaue Form der Ausgabe selbst bestimmen, die bei verschiedenen Textarten natürlich varriiert.

Fußnoten in mehrsprachigem Layout (F. Mittelbach, S. 657)

Bei mehrsprachigen Texten stößt der Anwender, der mehr als eine Fußnote zu setzen hat, unweigerlich auf Probleme. Die gängigen Methoden der Fußnotenverarbeitung sehen alle mehr oder weniger schön aus. Frank Mittelbach hat deswegen eine Umdefinition des Fußnotenmechanismus geschrieben, der die Anmerkungen an das Ende jeder Seite setzt. Diese Definition paßt auf jede Eingabe, die die twocolumn-Option benutzt. Der Code ist zusammen mit den Erklärungen in den Text eingearbeitet.

◇ Luzia Dietsche
Rechenzentrum der Universität
Im Neuenheimer Feld 293
D-6900 Heidelberg 1
Bitnet: X68@DHDURZ1

Letters

TUG Drug Bug

Jackie Damrau's LaTeX column in the June *TUGboat* left a funny taste in my (if not TeX's) mouth. What was truly being "shared" here? The simple macro is really just a Trojan horse for some offensive axe-grinding about marijuana and cocaine.

I enjoy TeX and *TUGboat* not just for their TeXnical EXcellence, but because it's fun to be a part of an international community of minds who meet electronically or in print. But the feeling of community depends on leaving other issues and "causes" outside the door. Please don't allow exaple-providers to abuse the channel as a sneaky means of getting political messages (in this case U.S. anti-drug propaganda) printed in *TUGboat*.

Respectfully yours,

Peter C. Akwai
Energy Transfer GmbH
Computer-Systeme & Beratung
Vor der Pforte 14
Postfach 50 11 18
6072 Dreieich, West Germany

Editor's note: We hear you. In the future, we will look more carefully at the content of examples as well as at their implementation.

Calendar

1990

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	7 – 11	Intensive Beginning/Intermed. TeX, University of Houston, Clear Lake, Texas

Providence College, Providence, Rhode Island

Jan	7 – 11	Intensive LaTeX
Jan	7 – 11	Intensive Beginning/Intermed. TeX

Jan	14 – 18	Advanced TeX/Macro Writing, California State University, Northridge, California
Jan	28 – Feb 1	Advanced TeX/Macro Writing, University of Maryland, College Park, Maryland
Feb	19	***TUGboat* Volume 12, 1st regular issue:** Deadline for receipt of *technical* manuscripts.
Feb	20 – 22	10th annual meeting, "Deutschsprachige TeX-Interessenten"; DANTE e.V.: 4th meeting, Technical University of Vienna. For information, contact Dr. Hubert Partl (Bitnet: `Z3000PA@AWITUW01`) or DANTE e.V. (Bitnet: `DANTE@DHDURZ1`)
Mar	19	***TUGboat* Volume 12, 1st regular issue:** Deadline for receipt of news items, reports, etc.
Mar	25 – 29	Intensive LaTeX, Northeastern University, Boston, Massachusetts

University of Hawaii at Manoa

Mar	25 – 29	Intensive Beginning/Intermed. TeX
Mar	25 – 29	Advanced TeX/Macro Writing

Apr	2 – 5	RIAO Conference on Intelligent Text and Image Handling, Universidad Autónoma de Barcelona, Spain. For information, contact (in the U.S.) RIAO '91, Center for the Advanced Study of Information Systems, Inc. (CASIS), Ms. M.-T. Maurice, 220 East 72nd Street #10F, New York, NY 10021; (in Europe) CID, 36 bis rue Ballu, F-75009 Paris, France.

TUG91 Conference
Dedham, Massachusetts (suburban Boston)

Jul	15 – 18	**TUG's 12th Annual Meeting**

Jul	25	***TUGboat* Volume 12, Proceedings issue:** Deadline for receipt of news items, reports (tentative).
Aug	11	***TUGboat* Volume 12, 2nd regular issue:** Deadline for receipt of *technical* manuscripts (tentative).
Sep	10	***TUGboat* Volume 12, 2nd regular issue:** Deadline for receipt of news items, reports (tentative).
Sep	23 – 25	**6th European TeX Conference** Paris, France. (See page 667.)
Sep	26	GUTenberg'91 Congress, Paris, France. "Technical and scientific edition". (See page 667.)
Oct	15 – 16	RIDT 91, The second international workshop on raster imaging and digital typography, Boston, Massachusetts. (See page 668.)
Dec	4	Course about fonts, Paris, France. Sponsored by GUTenberg. For information, contact Jacques André (`jandre@irisa.irisa.fr`).

For additional information on the events listed above, contact the TUG office (401-751-7760) unless otherwise noted.

Status as of 20 October 1990

Tenth Meeting of the German Speaking TEX Users in Vienna First Announcement and Call for Papers

Hubert Partl

The tenth meeting of the German speaking TEX users group DANTE will be held

- from February 20 to 22, 1991

at the Technical University of Vienna (Austria). Wednesday afternoon will be reserved for the general assembly of DANTE members only, whereas to the talks, discussions, and workshops on Thursday and Friday, attendees from all European and non-European countries are welcome, too.

The official conference language will be German, but talks may also be presented in English. Proposed topics include all aspects of TEX, LATEX, METAFONT, SGML and, of course, European language support.

The technical program will be supplemented by several non-technical events like a performance of Mozart's "Magic Flute" by the Vienna State Opera, an evening at a typical Viennese "Heurigen" wine restaurant, and a reception in the City Hall.

- If you want to present a paper or to exhibit a product, please tell us **before December 14, 1990,** to the address below.

All DANTE members will be invited individually at the end of this year.

- All non-members who would also like to attend the meeting or to receive more information about it, should tell us before the end of December.

The organiser's postal and electronic addresses are:

Dr. Hubert Partl
EDP Center of the
Technical University of Vienna
Wiedner Hauptstraße 8–10
A-1040 Wien, Austria

e-mail: `z3000pa@awituw01.bitnet`

Alternatively, you may also contact the appropriate TUG vice president, Joachim Lammarsch, in Heidelberg.

TEX91/Congres GUTenberg'91

Olivier Nicole
Secretary, GUTenberg

TEX91

The 6th European TEX Conference will be held in Paris (France), 23–25 September 1991. It will be organized by GUTenberg. Suggested topics are

- Multi-lingual TEX
- Users groups
- Development around TEX, METAFONT, ...
- Merging of TEX with other systems and applications
- Typographic layout and design
- Hypertext and its relationship with TEX
- Document markup systems
- Font design
- Merging of TEX and graphics
- Specialists macros and their applications
- Networks related to TEX

Information can be obtained from

GUTenberg
6th European TEX Conference
B.P. 21
78354 JOUY en JOSAS cedex
France
Phone: +33 1 34 65 22 32
Fax: +33 1 34 65 20 51
E-mail: `gut@irisa.irisa.fr`

GUTenberg '91

The 1991 GUTenberg Congress will be held in Paris, 26 September 1991. The topic "Technical and Scientific Edition" has been selected as a theme.

The following topics are suggested for papers.

- Special fonts design
- Developments around TEX and LATEX
- TEX and graphics
- Rasterization and output
- Special macro-packages
- Aspects of French language

Information is available from the same source as for TEX91.

Short course on fonts

Gutenberg will also be organizing a short course on fonts, to be held in Paris, 4 December 1991. For information, contact (e-mail only):

`jandre@irisa.irisa.fr`

Call for papers: RIDT 91, The second international workshop on raster imaging and digital typography

Boston, Massachusetts, 15–16 October 1991

The workshop is sponsored by the University of Massachusetts at Boston and the IEEE Computer Society. In addition, the organizers are planning concurrent one-day tutorials on type design, led by Hans Meier and Kris Holmes, and on type rasterization, led by Jacobo Valdes, to be held before the conference.

Submission deadline: January 15, 1991

Submissions will be refereed by the program committee. Accepted papers will be collected in a proceedings that the organizers expect to be published by Cambridge University Press. Full papers in English on any of the following or related topics are welcomed:

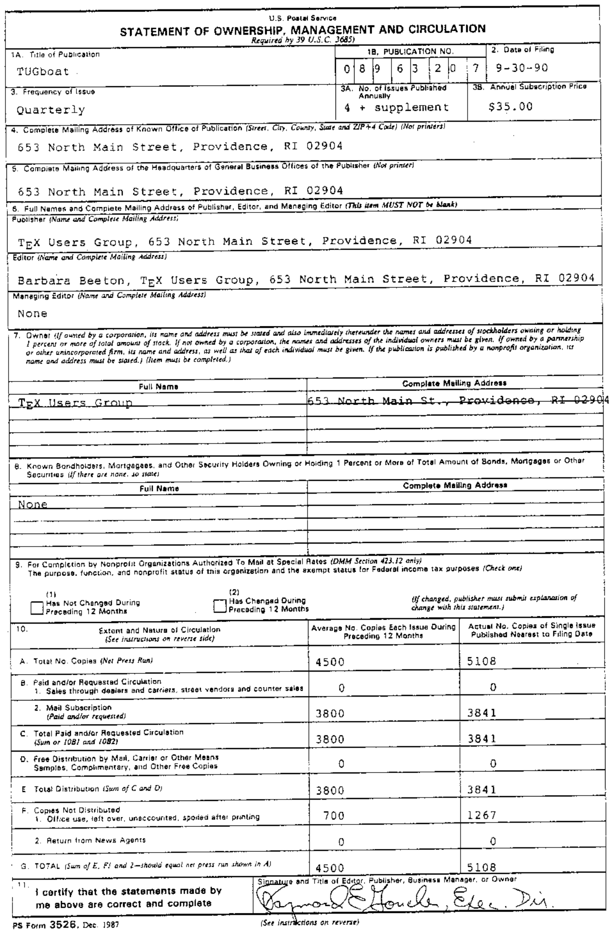
- measuring type quality
- character design, representation and transformation
- shape acquisition and manipulation
- color printing
- fast rasterization hardware
- applications of human vision science to type design
- character generation and recognition
- page description languages
- anti-aliasing
- digital halftone processing
- font representations for automatic scan conversion
- rasterization algorithms

If you wish to receive guidelines for authors or other electronic or paper mail about the conference, you should contact the chair:

Prof. Robert A. Morris, RIDT 90
Department of Mathematics and Computer Science
University of Massachusetts at Boston
Boston, MS 02125-3393
U.S.A.

telephone: (617) 287-6466
email: `ridt91-request@cs.umb.edu`

Other members of the Program Committee are: Roger Hersch, *EPFL* (Vice-chair); Debra Adams, *Xerox PARC*; Jacques André, *IRISA*; Patrick Beaudelaire, *DEC PRL*; Richard Beach, *Xerox PARC*; Charles Bigelow, *Stanford University*; Bruce Brown, *Oracle*; William Cowan, *University of Waterloo*; Andre Gürtler, *Schule für Gestaltung*; John Hobby, *Bell Laboratories*; Ernst Imhof, *URW*; Peter Karow, *URW*; Hideko Kunii, *Ricoh*; Yoshio Ohno, *Keio University*; Theo Pavlidis, *SUNY Stony Brook*; Stephen Schiller, *Adobe Systems*; Richard Southall, *Xerox Europarc*; Robert Ulichney, *DEC*; Jacobo Valdes, *Sun Microsystems*; Wang Xuan, *Peking University*.



U.S. Postal Service
STATEMENT OF OWNERSHIP, MANAGEMENT AND CIRCULATION
Required by 39 U.S.C. 3685)

1A. Title of Publication: TUGboat	1B. PUBLICATION NO.: 0 8 9 6 3 2 0 7	2. Date of Filing: 9-30-90	
3. Frequency of Issue: Quarterly	3A. No. of Issues Published Annually: 4 + supplement	3B. Annual Subscription Price: $35.00	

4. Complete Mailing Address of Known Office of Publication *(Street, City, County, State and ZIP+4 Code) (Not printers)*
653 North Main Street, Providence, RI 02904

5. Complete Mailing Address of the Headquarters of General Business Offices of the Publisher *(Not printer)*
653 North Main Street, Providence, RI 02904

6. Full Names and Complete Mailing Address of Publisher, Editor, and Managing Editor *(This item MUST NOT be blank)*
Publisher *(Name and Complete Mailing Address)*
TEX Users Group, 653 North Main Street, Providence, RI 02904
Editor *(Name and Complete Mailing Address)*
Barbara Beeton, TEX Users Group, 653 North Main Street, Providence, RI 02904
Managing Editor *(Name and Complete Mailing Address)*
None

7. Owner *(If owned by a corporation, its name and address must be stated and also immediately thereunder the names and addresses of stockholders owning or holding 1 percent or more of total amount of stock. If not owned by a corporation, the names and addresses of the individual owners must be given. If owned by a partnership or other unincorporated firm, its name and address, as well as that of each individual must be given. If the publication is published by a nonprofit organization, its name and address must be stated.) (Item must be completed.)*

Full Name	Complete Mailing Address
TEX Users Group	653 North Main St., Providence, RI 02904

8. Known Bondholders, Mortgagees, and Other Security Holders Owning or Holding 1 Percent or More of Total Amount of Bonds, Mortgages or Other Securities *(If there are none, so state)*

Full Name	Complete Mailing Address
None	

9. For Completion by Nonprofit Organizations Authorized To Mail at Special Rates *(DMM Section 423.12 only)*
The purpose, function, and nonprofit status of this organization and the exempt status for Federal income tax purposes *(Check one)*
(1) ☐ Has Not Changed During Preceding 12 Months (2) ☐ Has Changed During Preceding 12 Months *(If changed, publisher must submit explanation of change with this statement.)*

10. Extent and Nature of Circulation *(See instructions on reverse side)*	Average No. Copies Each Issue During Preceding 12 Months	Actual No. Copies of Single Issue Published Nearest to Filing Date
A. Total No. Copies *(Net Press Run)*	4500	5108
B. Paid and/or Requested Circulation 1. Sales through dealers and carriers, street vendors and counter sales	0	0
2. Mail Subscription *(Paid and/or requested)*	3800	3841
C. Total Paid and/or Requested Circulation *(Sum or 10B1 and 10B2)*	3800	3841
D. Free Distribution by Mail, Carrier or Other Means Samples, Complimentary, and Other Free Copies	0	0
E Total Distribution *(Sum of C and D)*	3800	3841
F. Copies Not Distributed 1. Office use, left over, unaccounted, spoiled after printing	700	1267
2. Return from News Agents	0	0
G. TOTAL *(Sum of E, F1 and 2—should equal net press run shown in A)*	4500	5108

11. **I certify that the statements made by me above are correct and complete**
Signature and Title of Editor, Publisher, Business Manager, or Owner
[illegible], Exec. Dir.

PS Form 3526, Dec. 1987 *(See instructions on reverse)*

BYLAWS of the TEX Users Group ("TUG")

Article I
PURPOSES, POWERS AND NON-PROFIT STATUS

Section 1. Purposes. The TEX Users Group (the "Corporation") has been formed exclusively for charitable, educational and scientific purposes as such terms are defined in Section 501(c)(3) of the Internal Revenue Code of 1986, or the corresponding provision of any future United States internal revenue law (hereinafter the Internal Revenue Code of 1986), and specifically to identify, develop, operate, fund, support, promote and encourage charitable, educational and scientific programs and projects which will stimulate those who have an interest in systems for typesetting technical text and font design; to exchange information of same and associated use of computer peripheral equipment; to establish channels to facilitate the exchange of macro packages, etc., through publications and otherwise; and to develop, implement and sponsor educational programs, seminars and conferences in connection with the foregoing and for any lawful purpose or purposes permitted under the Rhode Island Non-profit Corporation Act.

Section 2. Powers. The Corporation shall have the power, directly or indirectly, either alone or in conjunction or cooperation with others, to do any and all lawful acts and things and to engage in any and all lawful activities which may be necessary, or convenient to effect any or all of the purposes for which the Corporation is organized, and to aid or assist other organizations whose activities are such as to further accomplish, foster, or attain any of such purposes. The power of the Corporation shall include, but not be limited to, the acceptance of contributions in cash, in kind or otherwise from both the public and private sectors. Notwithstanding anything herein to the contrary, the Corporation shall exercise its powers only in furtherance of exempt purposes as such terms are defined in Section 501(c)(3) of the Internal Revenue Code of 1986 and the regulations from time to time promulgated thereunder.

Section 3. Non-Profit Status. The Corporation shall be nonprofit and shall not have or issue shares of capital stock, and shall not declare or pay dividends. No part of the net income or profit of the Corporation shall inure to the benefit of any member, director, officer, or other individual, or to the benefit of any organization not qualified for tax exemption under Section 501(c)(3) of the Internal Revenue Code except as permitted by law. No substantial part of the activities of the Corporation shall be carrying on propaganda, or otherwise attempting to influence legislation (except as otherwise provided by Internal Revenue Code Section 501(h)), or participating in, or intervening in (including the publication or distribution of statements), any political campaign on behalf of any candidate for public office. Upon the dissolution of this organization, assets shall be distributed for one or more exempt purposes within the meaning of Section 501(c)(3) of the Internal Revenue Code or corresponding Section of any future Federal tax code, or shall be distributed to the Federal Government, or to a state or local government, for a public purpose.

Article II
OFFICES

The Corporation will have offices at such places both within and without the State of Rhode Island as may from time to time be determined by the board of directors.

Article III
MEMBERS

Section 1. Constitution. The members of the Corporation will be such persons, natural or legal, who will meet such qualifications and requirements (including without limitation payment of initiation fees and dues) as from time to time may be established by the board of directors. The board of directors will be the sole judge of the qualifications of the members and its determination as to whether a person is or is not a member will be final. The board of directors may, in its discretion, create different classifications of members and prescribe different rights, privileges, qualifications or requirements for each class.

Section 2. Place of Meetings. All annual meetings of the members and all special meetings of the members called by the president or the board of directors will be held at such place, either within or without the State of Rhode Island, as will be stated in the notice of meeting.

Section 3. Annual Meetings. Meetings of the members will be held in conjunction with TUG conferences. Such conferences will normally be held annually; otherwise, an annual meeting of the members will be held on the first Monday of August in each year if not a legal holiday in the place where it is to be held, and, if a legal holiday, then on the next day following which is not a legal holiday, beginning at 10:00 a.m. or at any other time designated in the notice of the meeting. At each annual meeting, the members will transact such business as may properly come before the meeting. In the event of the failure to hold said annual meeting at any time or for any cause, any and all business which might have been transacted at such meeting may be transacted at the next succeeding meeting, whether special or annual.

Section 4. Special Meetings. A special meeting of the members, for any purpose or purposes, may be called by the President or by the Board of Directors. Any such call will state the purpose or purposes of the proposed meeting.

Section 5. Notice of Meetings. Written notice of each annual or special meeting stating the place, day and hour of the meeting (and the purpose or purposes of any special meeting) will be given by or at the direction of the president, the secretary or the person or persons calling the meeting to each member entitled to vote at such meeting not less than ten nor more than sixty days before the meeting. Business transacted at any special meeting of members will be limited to the purposes stated in the notice of the meeting or any written waiver thereof.

Section 6. Quorum. Fifty (50) members present in person, will constitute a quorum at all meetings of the members. If, however, such quorum will not be present at any such meeting, the members entitled to vote thereat will have power to adjourn the meeting from time to time, without notice other than announcement at the meeting, until a quorum will be present. At such adjourned meeting at which a quorum will be present any business may be transacted which might have been transacted at the meeting as originally called. If adjournment is for more than thirty days, a notice of the adjourned meeting will be given to each member entitled to vote at the meeting. When a quorum is present at any meeting, the vote of the holders of a majority of the votes entitled to be cast and present in person will decide any question brought before such meeting, unless the vote of a greater number is required by law. A voice vote will normally be considered sufficient for business actions. A show of hands may be requested when the outcome is in doubt.

Section 7. Access to Document. Nothing in these bylaws shall be construed to limit the access of TUG members to TUG documents. Members requesting copies of any TUG document may be charged a reasonable copying fee and members requesting publications or mailing lists presented to the public for sale may be charged the same fee as the general public. Members requesting copies of documents to be used in performance of TUG related duties may request that the copying fee be waived. TUG documents include, but are not limited to: contracts, Board minutes, Executive Committee minutes, Finance Committee minutes, office procedure manuals, IRS filings, and written communications from or to the TUG office. This section does not authorize the release of any information that federal or state law protects from disclosure.

Article IV
DIRECTORS

Section 1. Powers. The affairs of the Corporation will be managed by the board of directors.

Section 2. Number. The number of directors will be not more than thirty. Under very special circumstances, particularly deserving individuals may be designated as permanent honorary members of the Board, without vote, and without being included in the number of members specified in this section.

Section 3. Composition. The Board of Directors will consist of the Finance Committee, Site Coordinators, Wizards and other active members nominated by the Board of Directors.

Section 4. Honorary Members. The Grand Wizard, Donald E. Knuth, and the Wizard of Fonts, Hermann Zapf, are designated as permanent honorary members of the Board.

Section 5. Non-elected Vice Presidents. The leaders of other TeX user groups may be appointed to the

Board with the title of vice president. An increase in the number of members on the Board shall be made as appropriate.

Section 6. Election and Term. The first board of directors will be appointed by the incorporator. Thereafter, the Board of Directors shall have the power and responsibility to appoint and to remove its own members, except as provided in Section 8 of this Article, and each director appointed will hold office for a term of two (2) years and thereafter until his successor is appointed and qualified (unless there will be no successor as a result of a decrease in the number of the board of directors). Directors may be reappointed for successive terms. Directors need not be members of the Corporation or residents of the State of Rhode Island.

Section 7. Meetings. The board of directors may hold meetings, both regular and special, either within or without the State of Rhode Island. The first meeting of each newly elected board of directors will be held at such time and place as will be specified in a notice delivered as hereinafter provided for special meetings of the board of directors, or as will be specified in a written waiver signed by all of the directors. Regular meetings of the board of directors may be held without notice at such time and at such place as will from time to time be determined by the board of directors. Special meetings of the board of directors may be called by the president on two days' notice to each director, either personally or by mail or by telegram. Special meetings will be called by the president in like manner and on like notice on the written request of two directors. Meetings of the directors may be held by means of a telephone conference circuit and connection to such circuit will constitute presence at such meeting.

Section 8. Vacancies. Any vacancy occurring on the board of directors may be filled by the President. A director appointed to fill a vacancy will be appointed for the unexpired term of his or her predecessor in office. Any place on the board to be filled by reason of an increase in the number of directors may be filled by the President for a term of office continuing only until the next appointment of directors.

Section 9. Quorum. At all meetings of the board of directors, twenty-five (25%) percent of the number of directors fixed pursuant to Section 2 of this Article will constitute a quorum for the transaction of business, and the act of a majority of the directors present at a meeting at which a quorum is present will be the act of the board of directors, unless the act of a greater number is required by the Rhode Island non-profit corporation act or by the articles of incorporation.

Section 10. Directors' Consent Vote. Any action required or permitted to be taken at a meeting of the board of directors or of any committee thereof may be taken without a meeting if a consent in writing, setting forth the action so taken, will be signed by two-thirds of all directors or two-thirds of all the members of such committee, as the case may be. Members may use standard mail, electronic mail, or facsimile to cast a written vote. However, such action shall not be effective until the entire board or committee is notified by standard mail of the names of the members voting in favor of the action.

Section 11. Committees of Directors. The board of directors may, by resolution adopted by a majority of the board, designate one or more committees, including an executive committee, each committee to consist of two or more directors appointed by the board. The board may appoint one or more directors as alternate members of any committee, who may replace any absent or disqualified member at any meeting of the committee. Except as otherwise provided by the Rhode Island non-profit corporation act or these bylaws, any such committee, to the extent provided in the resolution, will have and may exercise all the authority of the board of directors; provided, however, that in the absence or disqualification of any member of such committee or committees, the member or members thereof present at any meeting and not disqualified from voting, whether or not he or she or they constitute a quorum, may unanimously appoint another member of the board of directors to act at the meeting in the place of any such absent or disqualified member. Such committee or committees will have such name or names as may be determined from time to time by resolution adopted by the board of directors. Each committee will keep regular minutes of its proceedings and report the same to the board of directors when required.

Section 12. Site Coordinator. The Site Coordinator will provide coordination for information about TEX, Metafont and other systems for typesetting technical text or font design with respect to a specific computer architecture and will provide technical direction for the future growth of TEX, Metafont, and/or other systems for typesetting technical type or font design.

ARTICLE V
COMMITTEES

Section 1. Executive Committee. There will be established an Executive Committee which will consist of the President, the elected Vice President, the Secretary and the Treasurer of the Corporation. It will be the responsibility of the Executive Committee to adopt interim procedures and policies when necessary on behalf of the Corporation, subject to the ultimate approval of the Board of Directors. The Executive Committee will have authority over all personnel matters of the Corporation and will report to the Board.

Section 2. Finance Committee. There will be established a Finance Committee which will consist of the Executive Committee and two other members qualified in financial affairs chosen by the Board of Directors. In order to ensure continuity, the outgoing President will remain a member of the Finance Committee for one year following the expiration of the term as President. Newly elected officers will join the Finance Committee between the time of election and the time at which they assume office (midnight, December 31). The Finance Committee will be responsible for supervising the management of all funds of the Corporation.

Section 3. Planning Committee. There will be established a Planning Committee responsible for establishing, with approval by the Board, TUG's strategic goals for recommending to the Board a three- (or more) year strategic plan to implement these goals. Members of the Planning Committee will be appointed by the President with the approval of the Board.

Section 4. Nominating Committee. Prior to the annual meeting, a Nominating Committee will be appointed by the Board for the purpose of suggesting candidates to fill those offices. This committee shall nominate at least one member to fill each office up for election.

Section 5. Ad Hoc Committees. The Board of Directors may from time to time, by resolution adopted by a majority of the Board, appoint one or more Ad Hoc Committees to perform such functions as may be designated in said resolution.

Article VI
NOTICES

Section 1. How Delivered. Whenever under the provisions of the Rhode Island non-profit corporation act or of the articles of incorporation or of these bylaws written notice is required to be given to any person, such notice may be given by mail, addressed to such person at his or her address as it appears in the records of the Corporation, with postage thereon prepaid, and such notice will be deemed to be delivered, if mailed, at the time when the same will be deposited in the United States mail. Notice may also be given by telegram or personally to any director.

Section 2. Waivers of Notice. Whenever any notice is required to be given under the provisions of the Rhode Island non-profit corporation act or the articles of incorporation or these bylaws, a waiver thereof in writing, signed by the person or persons entitled to such notice, whether before or after the time stated therein, will be deemed equivalent to the giving of such notice. Attendance of a person at a meeting will constitute a waiver of notice of such meeting, except when the person attends a meeting for the express purpose of objecting to the transaction of any business because the meeting is not lawfully called or convened.

Section 3. Specification of Business. Neither the business to be transacted at, nor the purpose of, any meeting of the members of the Corporation or of a committee of the board of directors of the Corporation need be specified in any written waiver of notice except as otherwise herein expressly provided.

Article VII
OFFICERS

Section 1. Number. The officers of the Corporation will be a president, a vice president, a secretary, and a treasurer. The board of directors may from time to time elect or appoint such other officers including more vice presidents and assistant officers, as it may deem necessary. Any two or more offices may be held by the same person with the exception of the offices of president and secretary.

Section 2. Eligibility for Nomination. Any active member in good standing may be nominated for office. Said member must accept the nomination before being placed on the ballot.

Section 3. Nomination Procedure. The Nominating Committee will nominate at least one member to fill each office up for election. In addition, any member may have his name placed in nomination by submitting a petition to the Nominating Committee at least thirty (30) days prior to the election signed by two (2) other members in good standing.

Section 4. Election and Term. The first officers of the Corporation will be appointed by its incorporator or by the initial board of directors. Thereafter, the officers will be elected by the general membership in accordance with the election procedures. The initial term of office of the officers shall be as follows: vice president, one year; president and secretary, two years; and treasurer, three years. Thereafter, each officer will be appointed or elected for a term not to exceed two years. The term of office will begin on January 1 of the year following election. Each officer will be elected to serve until his or her successor will have been elected and will have qualified or until his or her earlier death, resignation or removal as hereinafter provided. Any officer may be removed by the board of directors whenever in its judgment the best interests of the Corporation will be served thereby. Such removal will be without prejudice to the contract rights, if any, of the person so removed. Election or appointment of an officer will not of itself create contract rights.

Section 5. Election Procedures. All elections will be conducted by secret ballot. The candidate receiving the most votes will be elected. The requirement for a secret ballot may be waived by the President for an election for any office in which there is only one candidate.

Section 6. President. The President will preside at meetings of the General Membership, the Board of Directors and the Executive Committee.

Section 7. Vice President. The Vice President will serve in the absence of the President and will undertake other administrative duties as designated by the President.

Section 8. Secretary. The Secretary will maintain the records of the Corporation and see that all notices are duly given in accordance with the provisions of these Bylaws or as required by law. The Secretary will also conduct Corporate correspondence.

Section 9. Treasurer. The Treasurer will serve as chief financial officer and in general, will perform all of the duties incident to the office of Treasurer and such other duties as from time to time may be assigned to him by the President or Board of Directors.

Section 10. Vacancies. When an office becomes vacant for any reason, the President will appoint a member to serve out the remainder of that term. When the office of the President becomes vacant, the Vice President will become President for the remainder of the President's term and will then, as President, appoint a member to serve as Vice President.

Section 11. Signing of Instruments. All checks, drafts, orders, notes and other obligations of the Corporation for the payment of money, deeds, mortgages, leases, contracts, bonds and other corporate instruments may be signed by such officer or officers of the Corporation or by such other person or persons as may from time to time be designated by general or special vote of the board of directors.

Section 12. Voting of Securities. Except as the board of directors may generally or in particular cases otherwise specify, the president or the treasurer may on behalf of the Corporation vote or take any other action with respect to shares of stock or beneficial interest of any other corporation, or of any association, trust or firm, of which any securities are held by the Corporation, and may appoint any person or persons to act as proxy or attorney-in-fact for the Corporation, with or without power of substitution, at any meeting thereof.

Article VIII
EXECUTIVE DIRECTOR

Section 1. Duties. The Board of Directors shall select and employ an Executive Director who shall be responsible for the general administration of the Corporation's activities.

Section 2. Immediate Supervision. The Executive Director shall work under the immediate direction of the Executive Committee. The Executive Director shall attend meetings of the Executive Committee, the Finance Committee, and the Board of Directors, but shall not be a member of any of these bodies. The presiding officer of any of these meetings may request the absence of the Executive Director.

Article IX
SEAL

The corporate seal will have inscribed upon it the name of the Corporation and such other appropriate language as may be prescribed by the Rhode Island non-profit corporation act or from time to time by the board of directors.

Article X
FISCAL YEAR

The fiscal year of the Corporation will be determined by the board of directors and in the absence of such determination will be the calendar year.

Article XI
INDEMNIFICATION

Section 1. Agreement of Corporation. In order to induce the directors and officers of the Corporation to serve as such, the Corporation adopts this Article and agrees to provide the directors and officers of the Corporation with the benefits contemplated hereby.

Section 2. Acceptance of Director or Officer. This Article will apply, and the benefits hereof will be available, to each director and officer of the Corporation who executes and delivers to the Secretary of the Corporation a written statement to the effect that the director or officer accepts the provisions of this Article and agrees to abide by the terms contained herein.

Section 3. Definitions. As used herein, the following terms will have the following respective meanings:

"Covered Act" means any act or omission by the Indemnified Person in the Indemnified Person's official capacity with the Corporation and while serving as such or while serving at the request of the Corporation as a member of the governing body, officer, employee or agent of another corporation, partnership, joint venture, trust or other enterprise.

"Excluded Claim" has the meaning set forth in Paragraph 6, hereof.

"Expenses" means any reasonable expenses incurred by the Indemnified Person in connection with the defense of any claim made against the Indemnified Person for Covered Acts including, without being limited to, legal, accounting or investigative fees and expenses (including the expense of bonds necessary to pursue an appeal of an adverse judgment).

"Indemnified Person" means any director or officer of the Corporation who accepts election or appointment as a director or officer and agrees to serve as such in the manner provided in Paragraph 2 hereof.

"Loss" means any amount which the Indemnified Person is legally obligated to pay as a result of any claim made against the Indemnified Person for Covered Acts including, without being limited to, judgments for, and awards of, damages, amounts paid in settlement of any claim, any fine or penalty or, with respect to an employee benefit plan, any excise tax or penalty.

"Proceeding" means any threatened, pending or completed action, suit or proceeding, whether civil, criminal, administrative or investigative.

Section 4. Indemnification. Subject to the exclusions hereinafter set forth, the Corporation will indemnify the Indemnified Person against and hold the Indemnified Person harmless from any Loss or Expenses.

Section 5. Advance Payment of Expenses. The Corporation will pay the Expenses of the Indemnified Person in advance of the final disposition of any Proceeding except to the extent that the defense of a claim against the Indemnified Person is undertaken pursuant to any directors' and officers' liability insurance (or equivalent insurance known by another term) maintained by the Corporation. The advance payment of Expenses will be subject to the Indemnified Person's first agreeing in writing with the Corporation to repay the sums paid by it hereunder if it is thereafter determined that the Proceeding involved an Excluded Claim or that the Indemnified Person was otherwise not entitled to indemnity under these Bylaws.

Section 6. Exclusions. The Corporation will not be liable to pay any Loss or Expenses (an "Excluded Claim"):

(a) With respect to a Proceeding in which a final non-appealable judgment or other adjudication by a court of competent jurisdiction determines that the Indemnified Person is liable to the Corporation (as distinguished from being liable to a third party) for: (i) any breach of the Indemnified Person's duty of loyalty to the Corporation or its members; (ii) acts or omissions not in good faith or which involve intentional misconduct or knowing violation of law; or (iii) any transaction from which the Indemnified Person derived an improper personal benefit; or

(b) If a final, non-appealable judgment or other adjudication by a court of competent jurisdiction determines that such payment is unlawful.

Section 7. Notice to Corporation; Insurance. Promptly after receipt by the Indemnified Person of notice of the commencement of or the threat of commencement of any Proceeding, the Indemnified Person will, if indemnification with respect thereto may be sought from the Corporation under these Bylaws, notify the Corporation of the commencement thereof. Failure to promptly notify the

Corporation will not adversely affect the Indemnified Person's right to indemnification hereunder unless and only to the extent that the Corporation is materially prejudiced in its ability to defend against the Proceeding by reason of such failure. If, at the time of the receipt of such notice, the Corporation has any directors' and officers' liability insurance in effect, the Corporation will give prompt notice of the commencement of such Proceeding to the insurer in accordance with the procedures set forth in the policy or policies in favor of the Indemnified Person. The Corporation will thereafter take all the necessary or desirable action to cause such insurer to pay, on behalf of the Indemnified Person, all Loss and Expenses payable as a result of such Proceeding in accordance with the terms of such policies.

Section 8. Indemnification Procedures. (a) Payments on account of the Corporation's indemnity against Loss will be made by the Treasurer of the Corporation except if, in the specific case, a determination is made that the indemnification of the Indemnified Person is not proper in the circumstances because such Loss results from a claim which is an Excluded Claim. If the Corporation so determines that the Loss results from an Excluded Claim (although no such determination is required by the Corporation hereunder prior to payment of a Loss by the Treasurer), the determination shall be made:

(i) By the Board of Directors by a majority vote of a quorum consisting of directors not at the time parties to the Proceeding; or

(ii) If a quorum cannot be obtained for purposes of clause (i) of this subparagraph (a), then by a majority vote of a committee of the Board of Directors duly designated to act in the matter by a majority vote of the full Board (in which designation directors who are parties to the Proceeding may participate) consisting solely of three or more directors not at the time parties to the Proceeding; or

(iii) By independent legal counsel designated: (A) by the Board of Directors in the manner described in clause (i) of this subparagraph (a), or by a committee of the Board of Directors established in the manner described in clause (ii) of this subparagraph (a), or (B) if the requisite quorum of the full Board cannot be obtained therefor and a committee cannot be so established, by a majority vote of the full Board (in which designation directors who are parties to the Proceeding may participate). If made, any such determination permitted to be made by this subparagraph (a) will be made within 60 days of the Indemnified Person's written request for payment of a Loss.

(b) Payment of an Indemnified Person's Expenses in advance of the final disposition of any Proceeding will be made by the Treasurer of the Corporation except if, in the specific case, a determination is made pursuant to Paragraph 8(a) above that indemnification of the Indemnified Person is not proper in the circumstances because the Proceeding involved an Excluded Claim.

(c) The Corporation will have the power to purchase and maintain insurance on behalf of any Indemnified Person against liability asserted against him or her with respect to any Covered Act, whether or not the Corporation would have the power to indemnify such Indemnified Person against such liability under the provisions of this Article. The Corporation will be subrogated to the rights of such Indemnified Person to the extent that the Corporation has made any payments to such Indemnified Person in respect to any Loss or Expense as provided herein.

Section 9. Settlement. The Corporation will have no obligation to indemnify the Indemnified Person under this Article for any amounts paid in settlement of any Proceeding effected without the Corporation's prior written consent. The Corporation will not unreasonably withhold or delay its consent to any proposed settlement. If the Corporation so consents to the settlement of any Proceeding, or unreasonably withholds or delays such consent, it will be conclusively and irrebuttably presumed for all purposes that the Loss or Expense does not constitute an Excluded Claim. If the Corporation reasonably withholds its consent solely on the ground that the Proceeding constitutes an Excluded Claim, the Indemnified Person may accept the settlement without the consent of the Corporation, without prejudice to the Indemnified Person's rights to indemnification in the event the Corporation does not ultimately prevail on the issue of whether the Proceeding constitutes an Excluded Claim.

Section 10. Rights Not Exclusive. The rights provided hereunder will not be deemed exclusive of any other rights to which the Indemnified Person may be entitled under any agreement, vote of disinterested directors or otherwise, both as to action in the Indemnified Person's official capacity and as to action in any other capacity while holding such office, and will continue after the Indemnified Person ceases to serve the Corporation as an Indemnified Person.

Section 11. Enforcement. a) The Indemnified Person's right to indemnification hereunder will be enforceable by the Indemnified Person in any court of competent jurisdiction and will be enforceable notwithstanding that an adverse determination has been made as provided in Paragraph 8 hereof.

(b) In the event that any action is instituted by the Indemnified Person under these Bylaws, the Indemnified Person will be entitled to be paid all court costs and expenses, including reasonable attorneys' fees, incurred by the Indemnified Person with respect to such action, unless the court determines that each of the material assertions made by the Indemnified Person as a basis for such action was not made in good faith or was frivolous.

Section 12. Severability. If any provision of this Article is determined by a court to require the Corporation to perform or to fail to perform an act which is in violation of applicable law, this Article shall be limited or modified in its application to the minimum extent necessary to avoid a violation of law, and, as so limited or modified, this Article shall be enforceable in accordance with its terms.

Section 13. Successor and Assigns. The provisions of this Article will be (a) binding upon all successors and assigns of the Corporation (including any transferee of all or substantially all of its assets) and (b) binding on and inure to the benefit of the heirs, executors, administrators, and other personal representatives of the Indemnified Person.

Amendment. No amendment or termination of this Article will be effective as to an Indemnified Person without the prior written consent of that Indemnified Person and, in any event, will not be effective as to any Covered Act of the Indemnified Person occurring prior to the amendment or termination.

Article XII
AMENDMENTS

The power to alter, amend or repeal the bylaws or to adopt new bylaws will be vested in the board of directors by affirmative vote of the directors in the manner provided in these bylaws.

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*

CERN, *Geneva, Switzerland*

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*

Communications Security Establishment, Department of National Defence, *Ottawa, Ontario, Canada*

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*

Elsevier Science Publishers B.V., *Amsterdam, The Netherlands*

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*

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*

GTE Laboratories, *Waltham, Massachusetts*

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*

Hungarian Academy of Sciences, Computer and Automation Institute, *Budapest, Hungary*

IBM Corporation, Scientific Center, *Palo Alto, California*

Institute for Advanced Study, *Princeton, New Jersey*

Institute for Defense Analyses, Communications Research Division, *Princeton, New Jersey*

Intevep S. A., *Caracas, Venezuela*

Iowa State University, *Ames, Iowa*

The Library of Congress, *Washington D.C.*

Los Alamos National Laboratory, University of California, *Los Alamos, New Mexico*

Louisiana State University, *Baton Rouge, Louisiana*

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 TEX, 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*

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 LaTeX. (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.)

One article, by Yanai and Berry (p. 525) was prepared with *ditroff* and submitted as camera copy.

About two-fifths of the articles, and about half the pages in this issue are LaTeX. Articles in which no, or limited, TeX 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, in particular the Answers to Exercises for *TeX: The Program* (p. 499) used the experimental enhancement of the `plain` *TUGboat* macros that permits changing the number of columns in mid-page. When time permits, this will be cleaned up and made available via the archives.

The following articles were prepared using LaTeX.

- Nelson Beebe, *From the President*, page 485.
- *The future of TeX*, page 488.
- Nelson Beebe, *Comments on the future of TeX and* METAFONT, page 490.
- Jim Fox, *Webless literate programming*, page 511.
- Don Hosek, the Output device column, page 545, and two announcements (pages 570 and 578).
- Barbara Beeton, *A proto-TUG bibliograpy*, page 573.
- Joachim Lammarsch, IBM VM/CMS site report, page 578.
- Victor Eijkhout, all contributions, pages 572, 605, 613, 616.
- Lohn Lavagnino and Dominik Wujastyk, *Overview of* `EDMAC`, page 623.
- all items in the LaTeX section, pages 644 *ff.*
- Luzia Dietsche, German abstracts, page 663.

Output

The bulk of this issue was prepared on an IBM PC-compatible 386 using PC TeX and output on an APS-μ5 at the American Mathematical Society using resident CM fonts and additional downloadable fonts for special purposes.

Output for the article by Yanai and Berry (cited above) was prepared on a VariTyper VT600 and submitted as camera copy; some illustrations were prepared on an Apple LaserWriter (300 dpi) and on a Linotronic 100 (1270 dpi).

Figures for two articles were prepared by the authors on 300 dpi laser printers: the *Output routines* tutorial by David Salomon (p. 588), Apple LaserWriter, and *A TeX previewer* by Harold Stokes, HP LaserJet.

The output devices used to prepare the advertisements were not usually identified; anyone interested in determining how a particular ad was prepared should inquire of the advertiser.

Coming Next Issue

Babel

Johannes Braams describes Babel, a multilingual style-option system for use with LaTeX's standard document styles.

Network sources of TeXware

Peter Flynn provides an exhaustive list of network sites from which TeX and its relatives and friends can be retrieved by server or FTP.

Invisibility using virtual fonts

Sebastian Rahtz proposes an alternate method for generating "invisible" fonts as used by SliTeX. This method makes it possible to use the standard PostScript fonts in place of Computer Modern.

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 [ ]
Bus. [ ] Address: ______________________
____
____
____

Qty	1991 Membership/TUGboat Subscription (Jan.-Dec.)											Amount
	Ordinary: [ ] $45.00 Students: [ ] $35.00 (photocopy of student ID required)											
	TUGboat back volumes	1980	1981	1982	1983	1984	1985	1986	1987	1988	1989	1990	
	Circle volume(s) desired:	v. 1	v. 2	v. 3	v. 4	v. 5	v. 6	v. 7	v. 8	v. 9	v. 10	v. 11	
		$18	$50	$35	$35	$35	$50	$50	$50	$50	$75	$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): ______________________
____

Title: ______________________
Phone: ______________________
Network address: ______________________
[ ] Arpanet [ ] BITnet
[ ] CSnet [ ] uucp
[ ] JANET [ ] other ____________

Specific applications or reason for interest in TEX:

My installation can offer the following software or technical support to TUG:

Please list high-level TEX users at your site who would not mind being contacted for information; give name, address, and telephone.
____
____

Date: ______________________

Status of TEX: [ ] Under consideration
[ ] Being installed
[ ] Up and running since: ______
Approximate number of users: ______

Version of TEX:
[ ] Pascal
[ ] C
[ ] other (describe)
From whom obtained: ____________

Hardware on which TEX is used:

Computer(s)	Operating system(s)	Output device(s)

Revised 11/90

Each Institutional Member is entitled to:

- designate up to 7, 12 or 30 individuals to receive TUGboat subscriptions, depending on category of membership chosen; named individuals will be accorded full status as individual TUG members;
- reduced rates for TUG meetings/courses for *all* staff members, and for rental/purchase of videotapes;
- be acknowledged in every issue of TUGboat published during the membership year.

Instructions: Attach a list of the names and addresses of individuals to whom you would like TUGboat subscriptions mailed, to include answers to the questions on both side of this form–as approrpiate, in particular those regarding the status of TEX and the computer(s)/operating system(s) on which it runs or is being installed. (For IBM and VAX, especially, the operating system is more relevant than model.) It would be particularly useful if you could provide this information as it relates to each individual or group using the same hardware. Please make as many copies of this form as needed or contact the TUG office for additional copies.

- *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.

Institution/Organization: ____________________

Principal contact: ____________________

Phone: ____________

Qty	1991 Institutional Membership (Jan.–Dec.)												Amount
	Category A (incl. 7 subs.): educational $435; non-ed. $535; add'l subs. $40/ea.												
	Category B (incl. 12 subs.): educational $635; non-ed. $735; add'l subs. $40/ea.												
	Category C (incl. 30 subs.): educational $1260; non-ed. $1360; add'l subs. $35/ea.												
	TUGboat back volumes	1980	1981	1982	1983	1984	1985	1986	1987	1988	1989	1990	
	Circle volume(s) desired:	v. 1	v. 2	v. 3	v. 4	v. 5	v. 6	v. 7	v. 8	v. 9	v. 10	v. 11	
	Indiv. issues $18.00 ea.	$18	$50	$35	$35	$35	$50	$50	$50	$50	$75	$75	

Issues of TUGboat will be shipped by air service outside North America.

TOTAL ENCLOSED: ____________
(Prepayment in U.S. dollars required)

Membership List Information

Institution: ____________________

Principal contact: ____________________

Phone: ____________

Specific applications or reason for interest in TEX:

This installation can offer the following software or technical support to TUG:

Please list high-level TEX users at your site who would not mind being contacted for information; give name, address, and telephone.

____

Date: ____________

Status of TEX: [ ] Under consideration
[ ] Being installed
[ ] Up and running since: ______
Approximate number of users: ______

Version of TEX:
[ ] Pascal
[ ] C
[ ] other (describe)
From whom obtained: ____________

Hardware on which TEX is used:

Computer(s)	Operating system(s)	Output device(s)

Revised 11/89

TEX

T
H
E

A
R
B
O
R
T
E
X
T

W
A
Y

Another first from ArborText!

Virtual Fonts

ü Ü ö Ö ä Ä

We have added built-in support to

Preview and DVILASER

which makes it easy to use the new

multilingual features of TEX 3.0.

Future enhancements will provide

a standard set of virtual fonts based on

coding conventions adopted at TEX 90.

TEX 3.0 and support software is available now for DEC/Risc-Ultrix, Sun, Apollo, and HP9000 (300 & 400).
***μTEX* 3.0 is a newly enhanced version for the IBM PC.**

535 W. William St., Ann Arbor, MI 48103, (313) 996-3566, FAX (313) 996-3573

ARBORTEXT INC.

For TEX Users....

New Services and Prices from Computer Composition Corporation

We are pleased to announce the installation of several ***new output services*** now available to TEX users:

1. High Resolution Laser Imaging (1200 dpi) from *Postscript diskette files* created on either Mac- or PC-based systems.
2. High Resolution Laser Imaging (960 dpi) from *DVI magnetic tape or diskette files* using a variety of typefaces in addition to the Computer Modern typeface family.
3. High quality laser page proofs at 480 dpi.
4. **NEW PRICING** for high resolution laser imaging:
   a. From **Postscript text files** in volumes over 400 pages .... **$2.00 per page**
   b. From **Postscript text files** in volumes between 100 & 400 pages .... **$2.25 per page**
   c. From **Postscript text files** in volumes below 100 pages .. **$2.40 per page**
   d. From **DVI files** in volumes over 400 pages .... **$2.15 per page**
   e. From **DVI files** in volumes between 100 & 400 pages ...... **$2.30 per page**
   f. From **DVI files** in volumes below 100 pages .... **$2.45 per page**

   NOTE: DEDUCT $1.00 FROM THE ABOVE PRICES FOR HIGH QUALITY LASER PAGE PROOFS.
5. **All jobs shipped within 48 hours.**

Call or write for page samples or send us your file and we will image it on the output unit of your choice.

COMPUTER COMPOSITION CORPORATION
1401 West Girard Avenue • Madison Heights, MI 48071
(313) 545-4330 FAX (313) 544-1611
— *Since 1970* —

Publishing Companion® translates

WordPerfect

to

TEX or LATEX

IN ONE EASY STEP!

With **Publishing Companion**, you can publish documents using TEX or LATEX with **little or no TEX knowledge.** Your WordPerfect files are translated into TEX or LATEX files, so anyone using this simple word processor can immediately begin typesetting their own documents!

Publishing Companion translates EQUATIONS, FOOTNOTES, ENDNOTES, FONT STYLES, and much more!

Retail Price	$249.00
Academic Discount Price	$199.00

For more information or to place an order, call or write:

K-TALK® COMMUNICATIONS

30 West First Ave, Suite 100
Columbus, Ohio 43201
(614)294-3535
FAX (614)294-3704

TYPESET QUALITY WITH THE EASE OF WORD PROCESSING

TEX EDITION of MathEdit

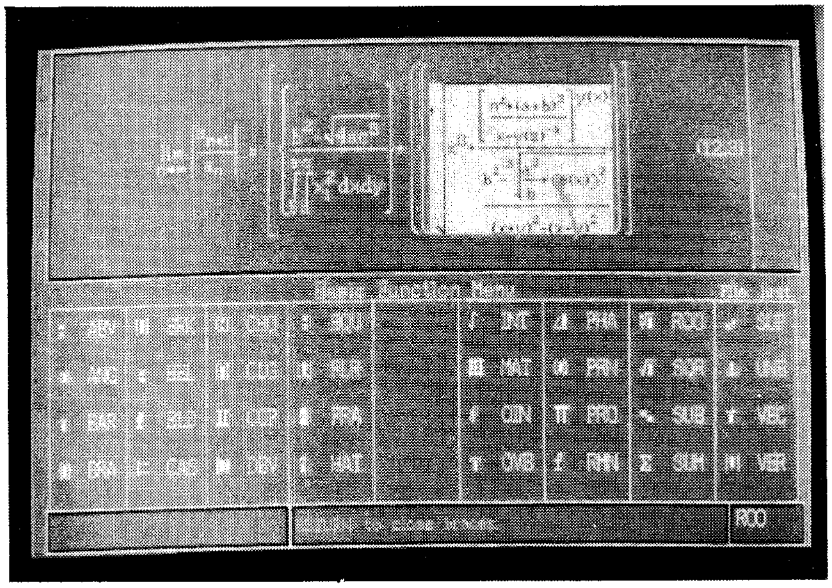
This powerful new equation editor can be used to create equations for TEX.
For IBM PC

MathEdit

* Easy to Use Version 2.0

* Menu driven so no codes need to be learned

* High-quality TEX printing



WYSIWYG –>
View your equation as you create it. Then insert into your TEX document with one command.

K·TALK® COMMUNICATIONS

30 West First Avenue
Columbus, Ohio 43201
(614) 294-3535
FAX (614) 294-3704

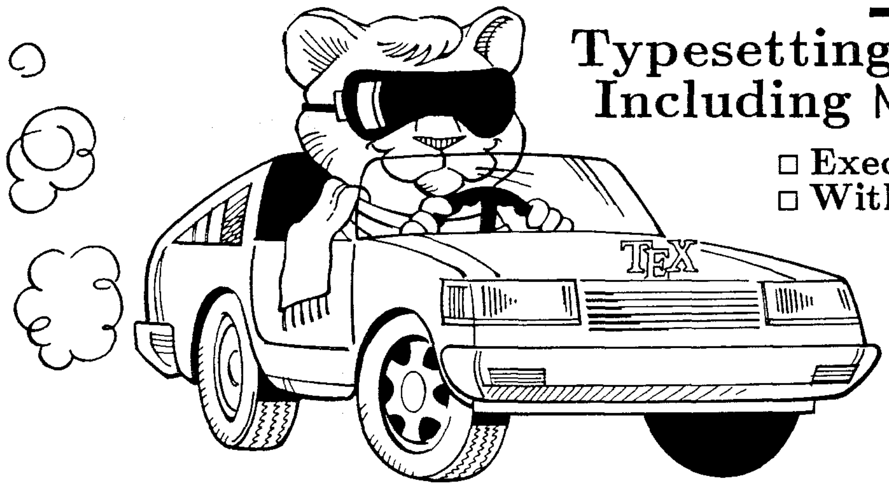
TEX Edition ONLY	$129.00
Professional Edition	$199.00

Shipping: $4 (U.S.A.), $25 (Canada), $35 (Overseas)
VISA, Mastercard and University and Government P.O.'s accepted.

TurboTEX

Typesetting Software Including METAFONT

□ **Executables $150**
□ **With source $300**



TEX 3.0 Update! Now Includes EGA/VGA Previewer!

TurboTEX Release 3.0 software brings you the latest TEX 3.0 and METAFONT 2.0 standards: preloaded plain TEX, LATEX, AMS-TEX and AMS-LATEX, and plain METAFONT interfaced to CGA/EGA/VGA/Hercules graphics; TRIP and TRAP certification; Computer Modern and LATEX 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 PICTEX and AMS-LATEX 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-TEX Fonts

TEX-compatible Bit-Mapped Fonts
Identical to
Adobe PostScript Typefaces

If you are hungry for new TEX fonts, here is a feast guaranteed to satisfy the biggest appetite! The AP-TEX fonts serve you a banquet of gourmet delights: 438 fonts covering 18 sizes of 35 styles, at a total price of $200. The AP-TEX fonts consist of `PK` and `TFM` files which are exact TEX-compatible equivalents (including "hinted" pixels) to the popular PostScript name-brand fonts shown at the right. Since they are directly compatible with any standard TEX implementation (including kerning and ligatures), you don't have to be a TEX expert to install or use them.

When ordering, specify resolution of 300 dpi (for laser printers), 180 dpi (for 24-pin dot matrix printers), or 118 dpi (for previewers). Each set is on ten 360 KB 5-1/4" PC floppy disks. The $200 price applies to the first set you order; order additional sets at other resolutions for $60 each. A 30-page user's guide fully explains how to install and use the fonts. Sizes included are 5, 6, 7, 8, 9, 10, 11, 12, 14.4, 17.3, 20.7, and 24.9 points; headline styles (equivalent to Times Roman, Helvetica, and Palatino, all in bold) also include sizes 29.9, 35.8, 43.0, 51.6, 61.9, and 74.3 points.

The Kinch Computer Company
PUBLISHERS OF TURBOTEX
501 South Meadow Street
Ithaca, New York 14850
Telephone (607) 273-0222
FAX (607) 273-0484

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. PostScript is a registered trademark of Adobe Systems Incorporated. The owners of these trademarks and Adobe Systems, Inc. are not the authors, publishers, or licensors of the AP-TEX fonts. Kinch Computer Company is the sole author of the AP-TEX fonts, and has operated independently of the trademark owners and Adobe Systems, Inc. in publishing this software. Any reference in the AP-TEX font software or in this advertisement to these trademarks is solely for software compatibility or product comparison. LaserJet and DeskJet are trademarks of Hewlett-Packard Corporation. 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 October 9, 1990.

Avant Garde Book
Avant Garde Book Oblique
Avant Garde Demibold
Avant Garde Demibold Oblique
Bookman Light
Bookman Light Italic
Bookman Demibold
Bookman Demibold Italic
Courier
Courier Oblique
Courier Bold
Courier Bold Oblique
Helvetica
Helvetica Oblique
Helvetica Bold
Helvetica Bold Oblique
Helvetica Narrow
Helvetica Narrow Oblique
Helvetica Narrow Bold
Helvetica Narrow Bold Oblique
Schoolbook New Century Roman
Schoolbook New Century Italic
Schoolbook New Century Bold
Schoolbook New Century Bold Italic
Palatino Roman
Palatino Italic
Palatino Bold
Palatino Bold Italic
Times Roman
Times Italic
Times Bold
Times Bold Italic
Zapf Chancery Medium Italic
Symbol ΔΦΓϑΛΠΘ
Zapf Dingbats ✂☞❑

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, LATEX, 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

Font Library Tapes (GF files)
____ 300 dpi VAX/VMS format
____ 300 dpi generic format
____ IBM 3820/3812 MVS format
____ IBM 3800 CMS format
____ IBM 4250 CMS format
____ IBM 3820/3812 CMS format

Tape prices: $92.00 for first tape, $72.00 for each additional tape. Postage: allow 2 lbs. for each tape.

Documents:	Price $	Weight	Quantity
TEXbook (vol. A) softcover	30.00	2	____
TEX: The Program (vol. B) hardcover	44.00	4	____
METAFONT book (vol. C) softcover	22.00	2	____
METAFONT: The Program (vol. D) hardcover	44.00	4	____
Computer Modern Typefaces (vol. E) hardcover	44.00	4	____
LATEX document preparation system	30.00	2	____
WEB language *	12.00	1	____
TEXware *	10.00	1	____
BibTEX *	10.00	1	____
Torture Test for TEX *	8.00	1	____
Torture Test for METAFONT *	8.00	1	____
METAFONTware *	15.00	1	____
Metamarks *	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.: 408-735-8006.

▼ ● ☆ ➧ □ ⇩ ★

$79

IS ALL IT WILL TAKE TO ADD A FEW OF THESE TO YOUR TEX DOCUMENTS

◇ ← ● ▲ ☆ ⬇ ○

With TEXPIC Graphics language*, you will have the tools to make graphics for your TEX documents. TEXPIC is now available from Bob Harris at:

MPI

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

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

The right of the University of Cambridge to print and sell all manner of books was granted by Henry VIII in 1534. The University has printed and published continuously since 1584

For details contact

TECHNICAL APPLICATIONS GROUP CAMBRIDGE UNIVERSITY PRESS
UNIVERSITY PRINTING HOUSE SHAFTESBURY ROAD CAMBRIDGE CB2 2BS ENGLAND

TELEPHONE (0223) 325070

VECTOR TEX

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 VTEX 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. New: scalable graphics.
- Discover the universe of MicroPress Font Library professional typefaces: not available from any other TEX vender.

List price $399 **Introductory offer $299**

Includes the VTEX typesetter (superset of TEX), 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 January 1, 1991. S/H add $5. COD add $5. WordPerfect Interface add $100. Site licenses available. Dealers' inquiries welcome. Professional typefaces available for older implementations of TEX.

MICRO PRESS

MicroPress Inc.

68-30 Harrow Street, Forest Hills, NY 11375
Tel: (718) 575-1816 Fax: (718) 575-8038

VTEX is a trademark of MicroPress Inc. Other Products mentioned are trademarks of their respective companies.

TEX 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.

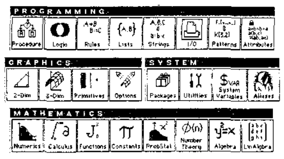
Mathematica Help

Variable Symbols, the company dedicated to giving you the tools to experience the full power of Mathematica, brings you 3 new ways to master Mathematica. When you want to be a more proficient Mathematica user, call us at 415-843-8701.

Mathematica Training
Hands-on workshops for 5–20 people at introductory, intermediate, and advanced levels taught by Variable Symbols staff, including Nancy Blachman, Cameron Smith, and Eran Yehudai. Our clients include Argonne, Boeing, Boston University, Genentech, Howard University, Lawrence Livermore, MITRE, Shell Development, Stanford University, Xerox, and Wolfram Research, Inc.

Mathematica Quick Reference
This reference defines all Mathematica built-in functions, options, aliases, and special forms; $9.95.

Mathematica Help Stack
HyperCard-based Macintosh help system that was developed by Robert Campbell categorizes built-in objects and provides visual representation of many functions for easy learning and reference; $89. Free demo disk available.



Variable Symbols, Inc.
P.O. Box 9013
Berkeley, CA 94709-0013
Telephone: 415-843-8701

TB10

Index of Advertisers

693	American Mathematical Society
683	ArborText
M-16	Blue Sky Research
691	Cambridge University Press
685	Computer Composition
690	DP Services
686, 687	K-Talk Communications
688, 689	Kinch Computer Company
692	MicroPress, Inc.
691	Micro Programs, Inc.
684	TEX Users Group
693	Variable Symbols, Inc.

Bugs in *Computers & Typesetting*

21 September 1990

This is a list of corrections made to *Computers & Typesetting*, Volumes A–E, since the last publication of the Errata and Changes list (25 March 90). Corrections 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.

Page A124, lines 18–21 (9/5/90)

Floating insertions can be accommodated as a special case of split insertions, by making each floating topinsert start with a small penalty, and by having zero as the associated `\floatingpenalty`; non-floating insertions like footnotes are accommodated by associating larger penalties with split insertions (see Appendix B).

Page A165, lines 2–3 (8/13/90)

Type the formula $\bar{\mathbf{x}}^{\rm T}\mathbf{Mx} = 0 \iff \mathbf{x} = \mathbf{0}$, using as few keystrokes as possible. (The first '0' is roman, the second is bold. The superscript 'T' is roman.)

Page A317, line 17 (5/17/90)

```
\pretolerance=9999 \tolerance=9999 \parindent=0pt
```

Page A321, lines 16–17 (8/13/90)

18.6. `$\bf\bar x^{\rm T}Mx={\rm0}\iff x=0$`. (If you typed a space between `\rm` and `0`, you wasted a keystroke; but don't feel guilty about it.)

Page Exiii, replacement for last four lines (4/30/90)

- "AMS Euler—A new typeface for mathematics" by Donald E. Knuth and Hermann Zapf, *Scholarly Publishing* **21** (1989), 131–157. *The story of a design project that helps bridge the gulf between mathematics and art.*

- "Meta-Marks: Preliminary studies for a Pandora's Box of shapes" by Neenie Billawala, Stanford Computer Science report 1259 (Stanford, California, July 1989), 132 pp. *Lavishly illustrated studies in parameter variation, leading to the design of a new typeface called Pandora.*

Changes to the Programs and Fonts

21 September 1990

TEX

Changes subsequent to errata publication, 25 March 90:

```
-----------Here I draw the line with respect to further changes

390. Uninitialized nullfont parameters (found by Lance Carnes, 11 May 90).
@x module 552
hyphen_char[null_font]:="-"; skew_char[null_font]:=-1;
@y
hyphen_char[null_font]:="-"; skew_char[null_font]:=-1;
bchar_label[null_font]:=non_address;
font_bchar[null_font]:=non_char; font_false_bchar[null_font]:=non_char;
@z

391. Disable \write{\the\prevgraf} (B. Jackowski, July 1990).
@x module 422
begin nest[nest_ptr]:=cur_list; p:=nest_ptr;
while abs(nest[p].mode_field)<>vmode do decr(p);
scanned_result(nest[p].pg_field)(int_val);
end
@y
if mode=0 then scanned_result(0)(int_val) {|prev_graf=0| within \.{\\write}}
else begin nest[nest_ptr]:=cur_list; p:=nest_ptr;
  while abs(nest[p].mode_field)<>vmode do decr(p);
  scanned_result(nest[p].pg_field)(int_val);
  end
@z

392. Report correct line number when buffer overflows (George Russell).
@x module 538
begin if input_ln(cur_file,false) then do_nothing;
firm_up_the_line;
if end_line_char_inactive then decr(limit)
else  buffer[limit]:=end_line_char;
first:=limit+1; loc:=start; line:=1;
@y
begin line:=1;
if input_ln(cur_file,false) then do_nothing;
firm_up_the_line;
if end_line_char_inactive then decr(limit)
else  buffer[limit]:=end_line_char;
first:=limit+1; loc:=start;
@z

393. (I sincerely hope that there won't be any more)
```

METAFONT

Changes since 25 March 1990.

```
-----------Here I draw the line with respect to further changes

555. Don't try system area if an area was given (see tex82.bug number 312;
found by Jonathan Kew, May 1990)
@x
  pack_file_name(cur_name,MF_area,cur_ext);
  if a_open_in(cur_file) then goto done;
@y
  if cur_area="" then
    begin pack_file_name(cur_name,MF_area,cur_ext);
    if a_open_in(cur_file) then goto done;
    end;
@z

556. Report correct line number when buffer overflows (CET, Jul 90).
@x module 794
begin if not input_ln(cur_file,false) then do_nothing;
firm_up_the_line;
buffer[limit]:="%"; first:=limit+1; loc:=start; line:=1;
@y
begin line:=1;
if input_ln(cur_file,false) then do_nothing;
firm_up_the_line;
buffer[limit]:="%"; first:=limit+1; loc:=start;
@z

557. (I sincerely hope that there won't be any more)
```

Computer Modern fonts

Changes since 25 March 1990.

No current changes.

TEX Users Group Membership List — Supplement

November 1990

This supplementary list, compiled on 15 October 1990, includes the names of all persons who have become members of TUG or whose addresses have changed since publication of the last membership list update, as of 15 May 1990 and bound into *TUGboat* Vol. 11, No. 2. Total membership: 145 institutional members and 3,789 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 TEX
- Uses to which TEX may be put, or a general indication of why the member is interested in TEX

CONTENTS

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.

Application to mail at second-class postage rate is pending at Providence, RI and additional mailing offices. Postmaster: Send address changes to the TEX Users Group, P. O. Box 9506, Providence, RI 02940, U.S.A.

Distributed with *TUGboat* Volume 11 (1990), No. 4. Published by

TEX Users Group
P. O. Box 9506
Providence, R.I. 02940-9506, U.S.A.

TEX Consulting and Production Services

North America

AMERICAN MATHEMATICAL SOCIETY
P. O. Box 6248, Providence, RI 02940; (401) 455-4060
Typesetting from DVI files on an Autologic APS Micro-5 or an Agfa Compugraphic 9600 (PostScript). Times Roman and Computer Modern fonts. Composition services for mathematical and technical books and journal production.

ANAGNOSTOPOULOS, Paul C.
433 Rutland Street, Carlisle, MA 01741; (508) 371-2316
Composition and typesetting of high-quality books and technical documents. Production using Computer Modern or any available PostScript fonts. Assistance with book design. I am a computer consultant with a Computer Science education.

ARBORTEXT, Inc.
535 W. William, Suite 300, Ann Arbor, MI 48103; (313) 996-3566
Typesetting from DVI files on an Autologic APS-5. Computer Modern and standard Autologic fonts. TEX installation and applications support. TEX-related software products.

ARCHETYPE PUBLISHING, Inc., Lori McWilliam Pickert
P. O. Box 6567, Champaign, IL 61821; (217) 359-8178
Experienced in producing and editing technical journals with TEX; complete book production from manuscript to camera-ready copy; TEX macro writing including complete macro packages; consulting.

THE BARTLETT PRESS, Inc., Frederick H. Bartlett
Harrison Towers, 6F, 575 Easton Avenue, Somerset, NJ 08873; (201) 745-9412
Vast experience: 100+ macro packages, over 30,000 pages published with our macros; over a decade's experience in all facets of publishing, both TEX and non-TEX; all services from copyediting and design to final mechanicals.

DOWNES, Michael
49 Weeks Street, North Smithfield, RI 02895; (401) 762-3715
Instruction in AMS-TEX, AMS-LATEX, plain TEX, and advanced macro writing. Custom documentstyles. Consulting: ■advanced mathematical typesetting topics; ■tuning mathematics fonts; ■ getting the most out of TEX in a production environment. Troubleshooting.

HOENIG, Alan
17 Bay Avenue, Huntington, NY 11743; (516) 385-0736
TEX typesetting services including complete book production; macro writing; individual and group TEX instruction.

KUMAR, Romesh
1549 Ceals Court, Naperville, IL 60565; (708) 972-4342
Beginners and intermediate group/individual instruction in TEX. Development of TEX macros for specific purposes. Using TEX with FORTRAN for custom-tailored software. Flexible hours, including evenings and weekends.

OGAWA, Arthur
920 Addison, Palo Alto, CA 94301; (415) 323-9624
Experienced in book production, macro packages, programming, and consultation. Complete book production from computer-readable copy to camera-ready copy.

QUIXOTE, Don Hosek
440F Grinnell, Claremont, CA 91711; (714) 625-0147
Complete line of TEX, LATEX, and METAFONT services including custom LATEX style files, complete book production from manuscript to camera-ready copy; custom font and logo design; installation of customized TEX environments; phone consulting service; database applications and more. Call for a free estimate.

RICHERT, Norman
1614 Loch Lake Drive, El Lago, TX 77586; (713) 326-2583
TEX macro consulting.

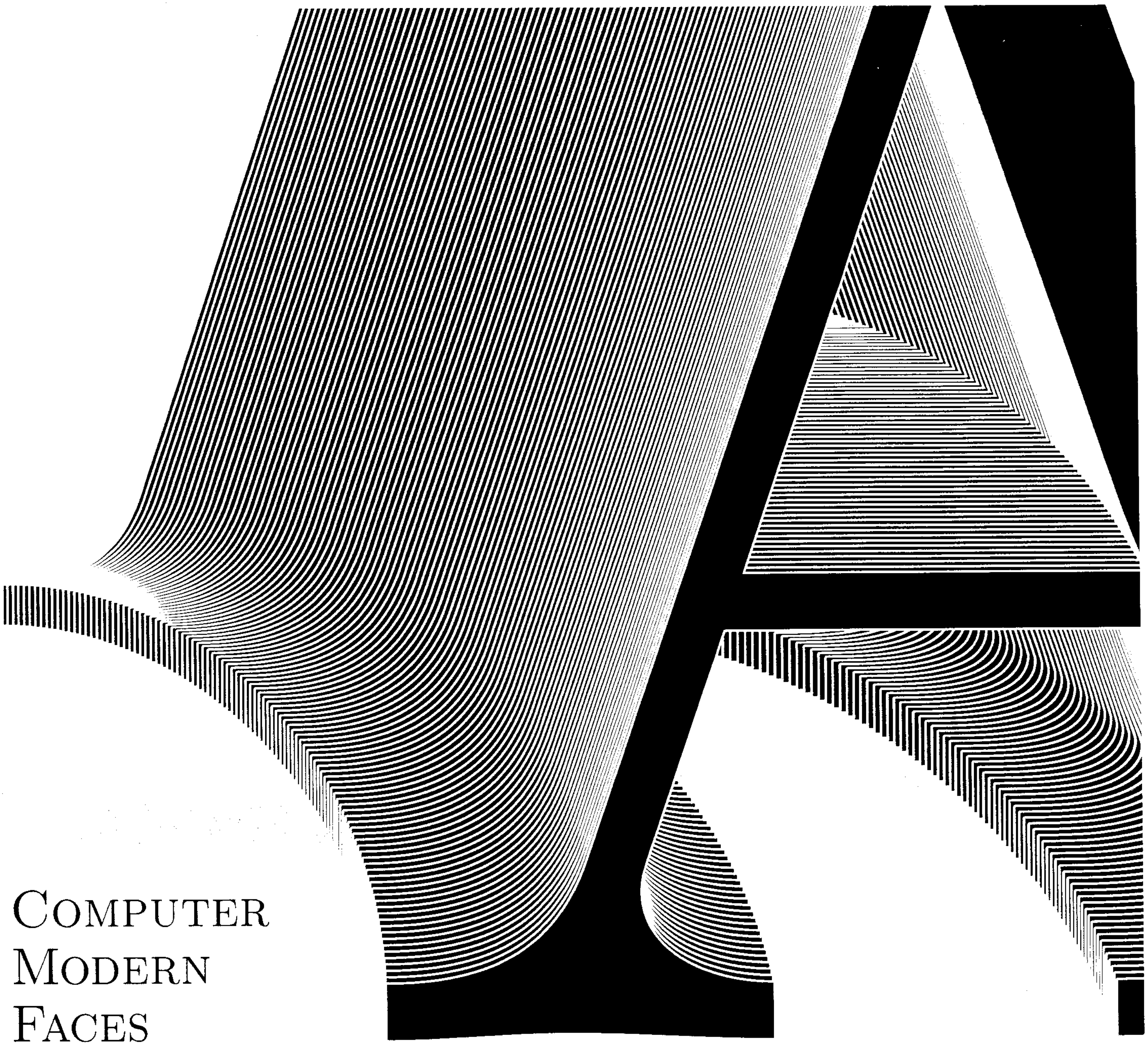
TEXNOLOGY, Inc., Amy Hendrickson
57 Longwood Ave., Brookline, MA 02146; (617) 738-8029.
TEX macro writing (author of MacroTEX); custom macros written to meet publisher's or designer's specifications; instruction.

Outside North America

TEXWORKS Pty. Ltd. (Alex Warman)
251 Moray Street, South Melbourne, 3205, Australia; 61 3 690 6023; Fax: 61 3 6994482
High resolution typesetting services from DVI files. Consulting and setup of production typesetting systems, especially automated typesetting from databases. TEX software for many computer platforms, training and support.

TREVORROW, Andrew
1-3-37 5th St., Habsiguda, Hyderabad 500007, India
TEX and PostScript programming services. Experience in VAX/VMS, UNIX and Macintosh environments. Author of DVItoVDU, PSPRINT and OzTEX. Prepared to travel anywhere!

Information about this service can be obtained from the TEX Users Group office, P. O. Box 9506, Providence, RI 02940, (401) 751-7760, Fax: (401) 751-1071.



Computer
Modern
Faces

Adobe
Type 1
PostScript
Fonts

Blue
Sky
Research

Forty faces of Computer Modern
designed by Donald Knuth
published in Adobe Type 1 format
compatible with
Adobe Type Manager
and all PostScript printers

$345.00 Educational $195.00
Macintosh or MS-DOS

Blue Sky Research
534 Southwest Third Avenue
Portland, Oregon 97204 USA
(800) 622-8398, (503) 222-9571
FAX (503) 222-1643