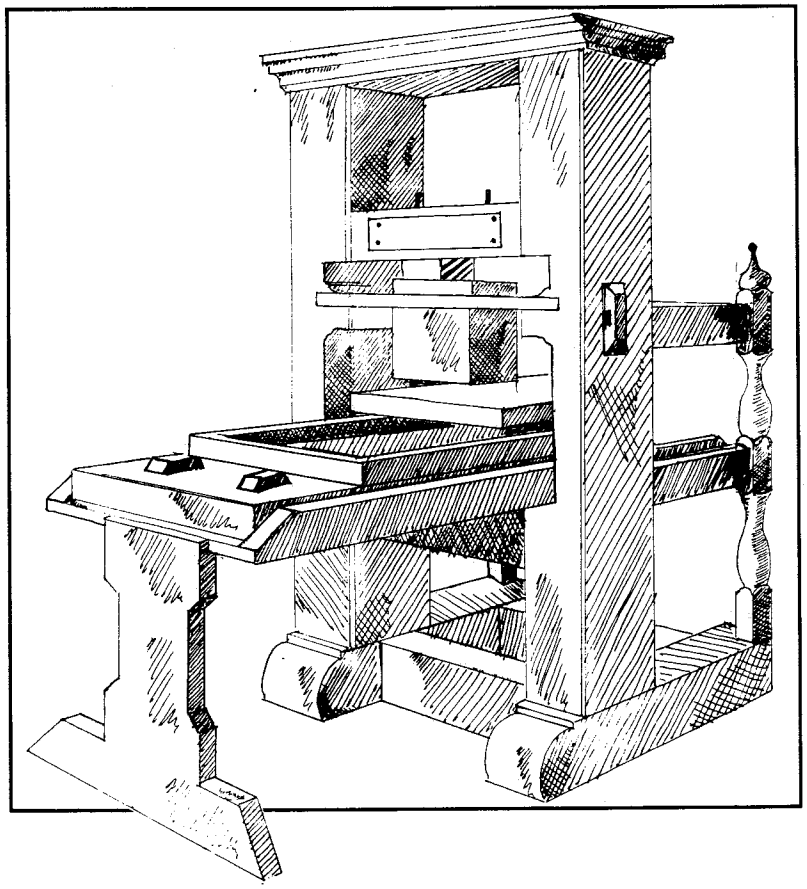
TUGBOAT

The Communications of the TEX Users Group



Volume 13, Number 1, April 1992

TEX Users Group

Memberships and Subscriptions

TUGboat (ISSN 0896-3207) is published four times a year plus one supplement by the TEX Users Group, 653 North Main Street, P. O. Box 9506, Providence, RI 02940, U.S.A.

1992 dues for individual members are as follows:

- Ordinary members: $60
- Students: $50

Membership in the TEX Users Group is for the calendar year, and includes all issues of *TUGboat* and *TEX and TUG News* for the year in which membership begins or is renewed. Individual membership is open only to named individuals, and carries with it such rights and responsibilities as voting in the annual election. A membership form is provided on page 112.

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

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

Institutional Membership

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

Board of Directors

See page 3 for addresses.

Addresses
General correspondence:
TEX Users Group
P. O. Box 9506
Providence, RI 02940

Payments:
TEX Users Group
P. O. Box 594
Providence, RI 02901

Parcel post, delivery services:
TEX Users Group
653 North Main Street
Providence, RI 02904

Telephone
401-751-7760

Fax
401-751-1071

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

Kerning should usually be completely invisible: the characters should look as if they have lived next to one another all their lives.

Ed Cleary, letter in
Seybold Report on Publishing Systems
(November 1989)

TUGBOAT

COMMUNICATIONS OF THE TEX USERS GROUP

EDITOR BARBARA BEETON

VOLUME 13, NUMBER 1 • APRIL 1992
PROVIDENCE • RHODE ISLAND • U.S.A.

TUGboat

During 1992, the communications of the TEX Users Group will be published in four issues. One issue (Vol. 13, No. 3) will contain the Proceedings of the 1992 TUG Annual Meeting.

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

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

Submitting Items for Publication

The deadline for submitting items for Vol. 13, No. 2, will have passed by the time this issue is mailed. The next regular issue will be Vol. 13, No. 4; deadlines are August 18, 1991, for technical items, and September 15, 1991, for reports and similar items. (Deadlines for future issues are listed in the Calendar, page 106.)

Manuscripts should be submitted to a member of the *TUGboat* Editorial Board. Articles of general interest, those not covered by any of the editorial departments listed, and all items submitted on magnetic media or as camera-ready copy should be addressed to the Editor, 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, are available "on all good archives". For authors who have no access to a network, they will be sent on request; please specify which is preferred. For instructions, write or call 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 Board

See page 3 for addresses.

Other TUG Publications

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

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

Trademarks

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

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

General Delivery

Editor's note: The reassignment of the Proceedings of the 1991 Annual Meeting to the final two issues last year excluded the outgoing President's farewell comments from their expected forum. Thus we begin this new year with remarks from two Presidents.

Prez sez

Living in interesting times

Malcolm Clark

Optimism is a necessary prerequisite for Board membership; a double dose is useful for the TUG President. It will not have escaped your notice that there is currently a world-wide recession, which shows no immediate sign of passing, despite the pronouncements of various election-sensitive politicians. TUG has not escaped the fallout of economic pressures: a number of effects have been experienced by us all. The most visible were the increase in dues; the restructuring of *TUGboat* 12 numbers 3 and 4 to be the proceedings issues; the cancelling of one issue of *TTN*; less visible has been the reduction in staff at the Providence office to only four, two of whom are working reduced hours.

Let's look on the bright side: *TUGboat* still managed to run to almost 600 pages; the Resource Directory was published; the final balance sheet will probably show that TUG just about broke even; *TUGboat* and *TTN* seem to be close to their production targets; the shortage of staff is making us look far more seriously at how best to use the vast pool of volunteers always lurking below the surface. I am also greatly encouraged by the cohesion shown at the recent Board meeting. The high degree of consensus and unanimity, directed towards common goals and objectives should help us all move towards a TUG which gives far more emphasis towards membership needs.

But of course, TUG is not the sole proprietor of TEX and the other TEX paraphernalia. A quick glance at the bits and pieces in the electronic archives, or the various digests and bulletin boards indicates that there is a vast, active, TEX-aware population out there. You might reasonably ask why they aren't all TUG members, but remember that TEX and its tools are not an end in themselves; they are merely software tools to aid document production. It might even be that the perceived missionary fervour dissuades many! But there is another key area where it is evident that TEX is becoming yet more respectable: there is now a large number of books, in several languages, on some aspects of TEX. Oddly, there are far fewer on LATEX (oddly, since LATEX is far more widely used — maybe it's intuitive). Publishers have clearly taken to heart the fact that the TEX and LATEX books have sold over 150,000 copies, and are seeking to exploit the demand.

Some thanks are due: a major debt of gratitude is due to Pierre MacKay and Tiina Modisett for the years of effort they put into `texhax`. The `texhax` electronic digest originated at Stanford University under the guidance of David Fuchs and was moderated for a time by Malcolm Brown. (I have much to thank Malcolm for: since Clark and Brown are apparently indistinguishable as surnames, I was able to bask in his glory for many years.) Then it was taken over by Pierre at the University of Washington. Peter Abbott (Aston electronic archive host, and currently chairman of the UK TEX users group) offered to administer the digest, using the tools developed by the similar (and monotonously regular) `uktex` digest, at nominal cost to TUG. This offer was gratefully accepted by the board. Users of the digest should notice no ill-effects. There have been some suggestions that a digest is no longer necessary, especially with the availability of `info-tex` and `comp.text.tex`. This is to misunderstand the function of these very different media. Those happily plugged into the electronic net sometimes overlook others' working patterns — I know that I find it far more convenient to find time to read a digest than to browse through an electronic conference or bulletin board.

I also have to express my thanks to those who are no longer on the TUG Board — Nelson Beebe, Lance Carnes, Bart Childs, John Crawford, Allen Dyer, David Fuchs, Regina Girouard, Dean Guenther, Hope Hamilton, Patrick Ion, David Kratzer, Pierre MacKay and Craig Platt (among them three past-Presidents). It is through the efforts of people like these that TUG has come as far as it has. They have provided the foundations on which we now stand.

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

President's Column

Nelson H. F. Beebe

1 Looking back

This is my last communication to you as President of the TEX Users Group for 1990–91. The past two years have seen considerable changes in the TUG office and the TUG Board, and change is often painful. With the new election procedures described in the pilot issue of *TEX and TUG News*, next year may see more changes, with possibly a very different membership on the Board. I wish the organization well, and hope that it can overcome the financial difficulties that it has suffered during the last three years, and move on to serve an ever-increasing number of TEX users. You can help: find some friends or colleagues who use TEX, but don't know about the TEX Users Group, explain the benefits of membership, and get them to join.

If I reflect on the last two and a half years, I confess that my biggest disappointments have been the turmoil in the Board, and the financial limitations that have prevented support of some important activities, particularly research into new directions in typography, such as the LATEX 3.0 redesign project, and beginning the groundwork for the design of possible successors to TEX and METAFONT.

Since the question of the names of descendents of TEX and METAFONT, in view of Don Knuth's wishes for immutable programs expressed in [13], has caused considerable confusion in the TEX community, I requested a clarification in a telephone conversation with Don a few days before writing this.

He made it quite clear that he wishes only that the names TEX and METAFONT be permanently bound to those programs as he wrote them; variants on the names, such as E-TEX [14], VTEX [19], or `nmf`, *are* acceptable to him. New names should be sufficiently different that there is no possibility of confusion with the old ones, even when accents are dropped, or letter case is mangled.

I must therefore retract my call [2] for a change in the name VTEX. So that no confusion remains as to my motives, I reiterate my position that experiments with new designs, like VTEX, MetaPost [11], Lilac [6], and the work of Luigi Semenzato and Edward Wang [16] are *vitally important* to the future of computer-assisted typesetting.

Don did express his hope that the architect(s) of any descendants of TEX and METAFONT will stand behind the new programs, as he has for his own creations.

2 Looking forward

One of my goals as President was to enhance access to TEX archives. The `tuglib` server at Utah has been operational for over a year, providing electronic mail access to archives for those who lack Internet `anonymous ftp` access. A preliminary description of the server appeared in UK TEXline at the Cork meeting last year, and is being reprinted by NTG. *TUGboat* publication of a revised version is scheduled for spring 1992.

The TUG office is collaborating with Jon Radel to provide distribution of TEXware on floppy disks.

The `tex-archive`, `tex-implementors`, and `tex-fonts` lists now provide a way for worldwide coordination of related activities. Maintenance of consistency in archives is a very large problem, because with several operating systems in use at archive sites, there is as yet no easy way for them to exchange files automatically. Eventually, we must find a solution, because the job is too large, and too important, to be relegated to fallible humans. The file headers described below provide a way to verify correctness and versions of files fetched from archives.

The Cork font standard should offer a solution to the use of 8-bit character sets with TEX 3.0. Work on extended Computer Modern fonts is nearly complete. The LATEX 3.0 development has had excellent ground work done, and I fully expect that the final product will be truly international.

TEX servers grow in number and size; the Aston server in the UK now even has its own Internet address. Within a few months, we hope to see it accessible from the Internet.

TUG membership now includes members from 51 countries, and over a dozen national/regional/language TEX user groups cater to the needs of specific segments of the TEX community.

TUG has launched a newsletter, TTN, to supplement *TUGboat*, and provide a forum for less technical communications.

TUG'92 will be held in Portland, Oregon, on July 27–30, 1992.

Malcolm Clark, the new TUG President for 1992, comes to us with ample TEX experience. He has written or edited at least three books with TEX, co-organized the EuroTEX88 meeting, edited and produced TEXline, chaired the UK-TEX group, and taught many courses about TEX and electronic document production. I'm confident he will do a fine job, and I'm sure you'll enjoy reading his contributions to *TUGboat*. Good luck, Malcolm.

3 News Items

3.1 EuroTEX91

The Sixth European TEX conference was held in Paris, France, on September 23–25, 1991, followed by the GUTenberg'91 meeting on September 26. About 120 people from at least 21 countries attended. The conference papers have already been published as numbers 10 and 11 of *Cahiers GUTenberg.*

Several papers dealt with the use of TEX in languages with accented characters, and working groups met daily for the polishing of the font standard proposed at the Cork meeting in 1990. We heard progress reports from representatives of each of the user groups, and I was pleased to hear that TEX is now being used for typesetting several journals in Eastern Europe.

The organizers did a fine job. Meals were provided one floor above the meeting hall. About ten workstations with Internet connections were freely available, allowing participants to login to their home machines to read mail and exchange software; this is a valuable service which I hope can become a tradition at future meetings. Simultaneous translations between French and English were provided. Participants were lodged in various Paris hotels, but thanks to the efficient Métro system, could quickly reach the meeting site.

EuroTEX92 is scheduled to be held in Prague, Czechoslovakia, in the month of September.

3.2 Project Gutenberg

I recently came across Project Gutenberg at the University of Illinois in Urbana/Champaign. Here are some comments from literature I obtained from them:

> The purpose of Project Gutenberg is to encourage the creation and distribution of English language electronic texts. We prefer the texts to be made available in pure ASCII formats so they would be most easily converted to use in various hardware and software. ... Our goal is to provide a collection of 10 000 of the most used books by the year 2001, and to reduce, and we do mean *reduce*, the effective costs to the user to a price of approximately one cent per book, plus the cost of media and of shipping and handling. Thus we hope the entire cost of libraries of this nature will be about $100 plus the price of the disks and CD ROMS and mailing. ... The easiest way for you to find out about Project Gutenberg is via subscription to the GUTNBERG listserver. You can do it by sending the following message to `listserv@uiucvmd.bitnet`:
>
> `SUB GUTNBERG your name`
>
> Your name must have at least two words. ... Please do not hesitate to ask for specific information so it is included in the GUTNBERG mailings. Please send these question messages separately from your subscription message.
>
> Michael S. Hart, Director, Project Gutenberg
> National Clearinghouse for Machine Readable Texts
> Bitnet: `Hart@uiucvmd`
> Internet: `Hart@vmd.cso.uiuc.edu`
>
> The GUTNBERG server is located at `gutnberg@uiucvmd.bitnet`. The Internet address is `gutnberg@vmd.cso.uiuc.edu`.

The 21-year old project is currently producing electronic versions of materials in the public domain, or for which copyright permission can be obtained, at the rate of one book per month. Plans are to double production yearly, and preparations are underway to produce a CD ROM of the current holdings. The collections are accessible for Internet `anonymous ftp` on several machines. The main archive is at `mrcnext.cso.uiuc.edu` in `~ftp/pub/etext`.

Through the GUTNBERG list, I uncovered another interesting effort at Georgetown University in Washington, DC, which reports

> Our project is involved in the cataloging projects around the world that are involved in the creations/storage/dissemination of electronic texts. So far, we have recorded the activities of over 320 projects around the world.

The catalog I received covers mainly the humanities, and only a small portion of the projects are accessible electronically.

3.3 Bibliography collections

The bibliography project mentioned several times in these columns continues to grow, with some collections receiving several updates a week. There are now over 51 000 lines in the bibliography files. A snapshot of the TEX-related material was published in the May 1991 TEX Users Group Resource Directory, which all of you should have seen by now.

Conversations that I had with representatives of several publishers at the TUG91 and EuroTEX91 meetings this year indicate that there may be another 1000–2000 books typeset by TEX that have not yet been included. I will continue efforts to get these incorporated, but the volume is large enough

that it will have to be from machine-readable material that can be manipulated into BibTeX form with the help of the wonderful `awk` language [1].

A recently added collection is the files `ep.*`, containing entries for papers from several recent electronic-publishing conferences; kudos to Karl Berry for initiating this one.

The collections are accessible via `anonymous ftp` to `math.utah.edu` from the directory `~ftp/pub/tex/bib`, and via electronic mail to `tuglib@math.utah.edu` with requests `help` to get started, and `send index from tex/bib` to find out what is there. Each BibTeX file has a corresponding LaTeX file to typeset the complete bibliography, and there are several supporting style files for BibTeX, LaTeX, and TeX.

3.4 Errata collections

I have recently established a new directory on `math.utah.edu` to hold errata files; any book in the bibliography collection is eligible for representation. The new book on LaTeX by Jane Hahn [10], and the new book by Raymond Seroul and Silvio Levy [18] (both cited in earlier columns) are covered there, as are some earlier books about TeX and LaTeX. Several of these errata and comment summaries can be typeset by LaTeX using a style file, `erratum.sty`, included in the collection.

You can find these files in `~ftp/pub/tex/errata` with `anonymous ftp`, or in `tex/errata` with the `tuglib` server.

3.5 Standard file headers

The file headers described in my editorial [3] a year ago are now being generated in new additions to the archives with the help of GNU Emacs Lisp code in the file `filehdr.el` stored with the bibliography collection. This code is quite general, and easily customized; it knows how to generate file headers for more than 110 different file types and over 60 different computer languages. Functions are provided for updating major and minor version numbers, dates, and checksums. The function `update-file-header-and-save` does all of those jobs, making it painless to maintain the headers after a file has been edited. Extended documentation will be available in LaTeXinfo format, which permits producing both a typeset manual, and on-line documentation.

Robert Solovay's `checksum` program in CWEB described in [4] is available for `anonymous ftp` from `math.utah.edu` from the directory `~ftp/pub/tex/pub/checksum`, and via electronic mail to `tuglib@math.utah.edu` with the request `send index from tex/pub/checksum` to find out what is there. `checksum` should compile and run on any system that has a C compiler, and an IBM PC executable is included in the distribution.

I encourage TeX archive sites, and authors, to incorporate these headers and checksums in files that are expected to be exchanged between systems. The American Mathematical Society has incorporated similar headers in the September 1991 release of $\mathcal{AMS}$-TeX 2.1. During the next year, I expect that tools will be developed to scan file directories, extracting information from the file headers to produce catalogs, abstracts, and version summaries.

Perhaps some clever programmer will volunteer to convert the approximately 1100 lines of code in `filehdr.el` to another editor language, such as that for `jove`, which runs on Apple and IBM personal computers, as well as UNIX and VAX VMS, so that the support for the file headers becomes readily available to a larger number of users.

3.6 Typesetting computerese

Peter Neumann's *Inside Risks* column in the Communications of the ACM [15] recently commented on the garbling of net addresses by LaTeX, `troff`, and hyphenation algorithms. Such problems have bothered this author, and I'm sure, the *TUGboat* editors.

Some time ago, I wrote a prototype of a TeX macro to typeset a string of characters in typewriter text, with discretionary breaks automatically inserted at punctuation characters, so that TeX could do a good job of line breaking without the tedium of the user having to supply explicit break points. I used it in the bibliography collection for file names, host names, and electronic mail addresses.

My implementation was not optimal, so after a redesign attempt that still did not meet my goals, I passed the problem off to Philip Taylor who kindly provided a robust solution. The macro is invoked as `\path|...|`; as with LaTeX's `\verb|...|`, you can pick the argument delimiters. However, the package goes further: you can specify the characters at which line breaks are permitted by saying something like `\discretionaries|-@%.!|`, which is a suitable set of break points for electronic mail addresses. This makes it easy to customize the line break points for particular applications. The default break set is all punctuation characters. The macros are available in the bibliography collection at Utah in the file `path.sty`.

3.7 New Books

As far as I am aware, as I write this, only one new book on TEX in English is in press, although I know of several others that are close to completion. Arvind Borde's *TEX by Example* [5] should be available from the publisher by the time you read this. I myself have not seen it yet.

Don Knuth informs me that the first book published by Soviet MIR Publishing using TEX will be a Russian translation of *Concrete Mathematics* [8], using special Concrete Cyrillic fonts designed with METAFONT especially for the book, following the style of Concrete Roman [12].

I recently obtained a copy of the book *Code Typographique* [7]. It is a comprehensive collection of rules for French typography, and for the typesetting of several foreign languages in French texts. It includes tables of diacritics for most European languages, plus hyphenation rules for English, German, Greek, Italian, Latin, Portuguese, and Spanish.

3.8 New TEX packages

TEXhax recently reported an implementation of a LATEX extension for Arabic typesetting [9]; it includes METAFONT sources and LATEX style files. This is a preliminary distribution from Prof. Klaus Lagally <lagally@informatik.uni-stuttgart.de>. The collection is stored on servers at ifi.informatik.uni-stuttgart.de and niord.shsu.edu, and may be expected to appear on others as well.

3.9 Electronic mail

Users of electronic mail have frequently experienced problems of corruption (particularly in TEX files), truncation, non-delivery, and the general inability to send 8-bit character data without special encoding. A June 1991 Internet Draft entitled *Mechanisms for Specifying and Describing the Format of Internet Message Bodies* can be obtained by sending an e-mail message with the text SENDME DRAFT.BODY* to d-vlserv@shsu.bitnet. It describes a proposal for extending electronic mail support to character sets other than US-ASCII, inclusion of binary data and image and audio fragments, and tagging of mail objects to convey type information, using a syntax compatible with SGML. As a draft, this document is circulated for comment only: in no way does it represent a commitment from the Internet authorities or network software developers. Nevertheless, I found particularly interesting its incorporation of a "text-plus" type which includes TEX, troff, and PostScript, and several proposed binary formats, one of which is TEX DVI. This may give us hope that by mid-decade, electronic mail may be much less troublesome for TEX users.

References

[1] Alfred V. Aho, Brian W. Kernighan, and Peter J. Weinberger. *The AWK Programming Language.* Prentice-Hall, 1988. ISBN 0-201-07981-X.

[2] Nelson H. F. Beebe. Comments on the future of TEX and METAFONT. *TUGboat*, 11(4):490–494, November 1990.

[3] Nelson H. F. Beebe. From the President. *TUGboat*, 11(4):485–487, November 1990.

[4] Nelson H. F. Beebe. President's introduction. *TUGboat*, 12(2):205–208, June 1991.

[5] Arvind Borde. *TEX by Example.* Academic Press, 1992. ISBN 0-12-117650-9.

[6] Kenneth P. Brooks. Lilac: A two-view document editor. *Computer*, 24(6):7–19, June 1991.

[7] Fédération C. G. C. de la Communication. *Code Typographique—Choix de règles à l'usage des auteurs et professionels du livre*, seizième edition, 1989.

[8] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics.* Addison-Wesley, 1989. ISBN 0-201-14236-8.

[9] George D. Greenwade. ArabTEX files available at FILESERV. *TEXhax*, 91(40), September 15 1991.

[10] Jane Hahn. *LATEX for Everyone.* Personal TEX Inc., 12 Madrona Street, Mill Valley, CA 94941, USA, 1991.

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

[12] Donald Knuth. Typesetting *Concrete Mathematics. TUGboat*, 10(1):31–36, April 1989.

[13] Donald E. Knuth. The future of TEX and METAFONT. *TUGboat*, 11(4):489, November 1990.

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

[15] Peter G. Neumann. Inside risks—expecting the unexpected Mayday! *Communications of the Association for Computing Machinery*, 34(5): 128, May 1991.

[16] Luigi Semenzato and Edward Wang. A Text Processing Language Should be First a Programming Language. *TUGboat*, 12(3):434–441, December 1991.

[17] Raymond Seroul. *Le petit Livre de TEX*. Inter-Editions, 1989. ISBN 2-7296-0233-X.

[18] Raymond Seroul and Silvio Levy. *A Beginner's Book of TEX*. Springer-Verlag, 1991. ISBN 0-387-97562-4, 3-540-97562-4. This is a translation and adaption by Silvio Levy of [17].

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

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

Editorial Comments

Barbara Beeton

It will not escape your notice that there has been a considerable delay between the previous regular issue and the present one, with the result that some items are no longer "fresh". This is regrettable, but I hope their usefulness is not affected by the delay.

Some of the news we have to offer is sad, but there are also some bright spots.

Cathy Booth, a remembrance

Cathy Booth, one of the most dedicated TEX supporters in the U.K., was defeated by cancer last July.

I remember meeting Cathy first at the TEX meeting in Strasbourg in 1986. She was cheery, outgoing, and always ready to help someone else.

With Malcolm Clark, Cathy organized the "TEXeter" meeting in 1988. For me, the local arrangements for this meeting were the most successful of all the EuroTEX meetings, with all the participants housed in a single Exeter University residence. Plenty of lounges and discussion areas and few distractions made it easy for everyone to really get to know one another. Cathy was the person to thank for this.

Our contact continued at TUG and other EuroTEX meetings, and in between, through electronic mail. It was a real shock to learn that she was ill, not from Cathy herself, but from a mutual friend.

I last saw Cathy in Cork, at EuroTEX'90. As always, she spread cheer and caring, even when she was obviously very tired and just keeping up was a great effort.

Her friends in the UKTEXug have established a fund in her memory, and the prize for the best paper at EuroTEX meetings has been named in her honor. All her many friends will miss her. I am glad to have had the opportunity to know her.

Sam Whidden, a remembrance

Another TEX stalwart lost to us was Sam Whidden. A more formal recollection follows this column. However, I can't omit saying what a good boss and friend he was for so many years. He helped mold the way my generation of AMS employees approach and solve problems. Sometimes he just didn't let on that he thought something couldn't be done, and was rewarded by seeing it done by his staff who didn't know any better.

The department Sam built was remarkably free of bureaucracy, and he always gave us opportunities to learn interesting new things. I hope that I pass on some of that enthusiasm to others. My world just isn't the same without Sam around.

Trip report: EuroTEX, GUTenberg'91

The sixth European TEX Conference took place on 23–25 September 1991 in Paris. Like previous editions, it was attended by a diverse collection of speakers and audience, on this occasion 121 people from 21 countries, including several in eastern Europe.

The spread of (LA)TEX in new geographic and language areas was a recurring theme throughout the conference. Reports were presented on activities and developments in Russia (and separately Siberia), Czechoslovakia, Poland, Hungary, Turkey, for African languages, for languages using Arabic scripts, and other topics related to linguistic and multilingual support. Several of the formal presentations on these topics appear in the proceedings, which form N° 10–11 of the *Cahiers GUTenberg*, distributed as part of the registration materials.[1]

Other presentations included updates on existing packages — Babel, MakeIndex, LAMS-TEX, and of course LATEX 3.0 — and reports on new work — database applications, SGML, windowing environments, tree structures, and color.

[1] See abstracts of this issue of the *Cahiers*, p. 101

Consistent with the interest in using the DC font arrangement agreed on in Cork, a lively BOF took place on the subject of 256-character math fonts; see below for more details. Other BOFs addressed LaTeX 3.0 and the future of TeX.

As has become the custom at EuroTeX meetings, a prize (a bottle of good Scotch) was awarded for the best paper. (With this year's presentation, this will become known as the Cathy Booth Prize.) The recipient of this year's award was Jiří Zlatuška, who spoke on automatic generation of fonts with accented letters, based on the existing Computer Modern fonts rather than on the extended layout.

By the end of the meeting, representatives of most of the European TeX groups had conferred and agreed, and it was announced that the 1992 EuroTeX meeting would be held in Prague.[2]

The following day, September 26, was devoted to the GUTenberg meeting. The program included several technical papers, panel discussions on matters of particular interest to French speakers, and GUTenberg business.

Throughout both meetings, the organizers made available terminals attached to an Internet connection, allowing participants to maintain connections to their home systems. Another welcome facility was simultaneous translation between French and English. The organizers deserve congratulations for a job well done.

256-character math fonts

The adoption of the 256-character DC fonts in Cork has addressed a number of problems in handling Western European languages that use the latin alphabet. However, some of the math capabilities of TeX have been disabled, and some symbols "orphaned" because of it.

At the request of Michael Ferguson, TUG's coordinator for multilingual activities, and now the chair of the TUG Technical Council, I undertook work on the creation of a compatible 256-character math font, and was able to make a preliminary report at a EuroTeX BOF in Paris last September.

The core principle expressed in this report was that `cmsy` and all "orphaned" `cm` symbols must be accommodated. Full upper- and lowercase greek alphabets are required, in both upright and italic postures. Blackboard bold is a strong candidate for inclusion, as are the most useful items from `msam`, `msbm`, `lasy`, `wasy`, and perhaps other existing metafonts. Suggestions and recommendations were solicited at the meeting.

I have since been in contact with Norbert Schwarz (who was largely responsible for the basic structure of the DC text fonts); he is constructing test versions of 256-character math fonts to begin experimentation. We are sharing the information that we have each collected, and I expect that Norbert will have a usable experimental layout by the time you read this. However, suggestions are still welcome; they should be accompanied, if possible, by supporting documentation of actual usage. Send them to me at the address printed at the end of this column, and I will make sure that they are forwarded to Norbert.

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

[2] An announcement for EuroTeX 92 can be found on p. 107

Samuel Blackwell Whidden, 1930-1991

Sam Whidden, one of the founders of TUG, and its treasurer from 1980 through 1987, died unexpectedly on October 29, 1991. At the time of his death, he was a member of TUG's Long Range Planning Committee.

Trained as an astronomer at Harvard College, Sam spent a couple of years at a hill station in India tracking Sputnik. After his return to the U.S., he obtained an M.B.A. degree from Harvard Business School and joined a small company that was developing computerized warehousing, shipping, and other services for publishers. These services included some experimental projects in text processing for the American Mathematical Society, and in April 1968, Sam came to work for AMS as founding Director of the Information Systems Development Department (ISD).

Sam's new department was charged principally with two tasks. The first was relatively ordinary: to develop in-house computer procedures to replace manual procedures and contracted computer services for the Society's business functions — accounting, sales, warehousing and shipping. The second was certainly more interesting, and perhaps more important: to continue investigations begun nearly a

Sam Whidden, at a lighter (than air) moment

decade earlier into computer-based typesetting and other experimental areas of text processing.

Sam was a champion of the idea that the Society could accomplish the composition of its books and journals on its own computer. This was the logical outgrowth of several projects that had been adopted by ISD on his arrival. He oversaw investigations into several promising technologies and helped design and conduct the pilot projects for math composition that brought first the Science Typographers system and later TEX into everyday use at AMS.

The Society's annual Gibbs Lecture was presented in 1978 by Donald Knuth. Richard Palais, then Chair of the Society's Board of Trustees (and later the first Chair of TUG), drew Sam's attention to this work, and encouraged an investigation. The investigation led to sending a small group of mathematicians and Society employees to Stanford in the summer of 1979. They were instructed to learn TEX, develop some tools, and bring back a working system that AMS could use for all its publication needs, to replace the two widely differing systems in use at the time.

When it became clear that TEX wasn't exactly going to be a "black box" with strong vendor support, Sam was instrumental in arranging with the Stanford TEX Project a meeting, held in February 1980, of existing TEX users and others interested in computer typography for the purpose of organizing a users group. From this meeting emerged TUG. Richard Palais was the first Chairman of the governing body, then called the Steering Committee, and Sam became the first Treasurer.

For the first year or so, the TUG "office" was really a minor activity under Sam's direction at AMS. As it grew, however, more support was required, and Sam's administrative assistant, Ray Goucher, left his position at the Society to become TUG Business Manager. Sam continued to watch over the operation, both as Treasurer and in an advisory capacity, until more space was needed than was available at the AMS offices, and the TUG office moved to its own quarters.

Sam spread the word about TEX and TUG in other directions too. He was an active participant in DECUS, the Digital Equipment Corp. Users Society, chairing the Languages & Tools SIG for three years and making his presence felt in a number of other important areas. He encouraged the use of TEX for preparing DECUS documents, and assigned members of his staff to speak at DECUS about TEX and to develop macros for preparation of the DECUS proceedings. (These macros are still in use.) Sam's own contribution was a set of LATEX macros for the "Sessions-at-a-Glance", a compact room-schedule chart that is part of every DECUS program.

In recognition of his work, Sam was the recipient of several of DECUS's highest awards. The 1991 Fall Symposium in Anaheim was dedicated to him.

Sam's final legacy to TUG was his outline for the direction of the Long Range Planning Committee. As long-time Treasurer, he was concerned for the financial health of the organization, but also for its less tangible qualities, such as the sense of community among its members. His foresight and advocacy spurred the creation of the committee, and his participation in its deliberations are now beginning to bear fruit in the form of a plan that will be able to guide TUG in the years to come.

Sam is survived by his mother, a sister, four children, and two grandchildren. He also leaves behind many appreciative friends and colleagues.

Barbara Beeton
American Mathematical Society

Software

Inside Type & Set

Graham Asher

Abstract

Type & Set is a typesetting system consisting of TEX, several TEX macro packages, and a suite of C programs including a style sheet editor, an automatic page make-up system which replaces TEX's output mechanism, and a family of drivers. It solves many of the problems which make plain TEX difficult to use for commercial journal and book publishing. This article explains in detail how Type & Set works.

History of the project

Type & Set has been under development at Informat Computer Communications since February 1987. Informat is the software development and typesetting division of Current Science (formerly Gower Academic Journals), a publishing house, and because the two companies share the same premises we have had constant access to users and their suggestions and criticism. Some ideas in Type & Set are taken from an earlier (non-TEX) package of the same name which it has superseded. For these ideas (principally the style sheet hierarchy, the mark-up system and the input format for the table generator) I am indebted to Mr. A. Harris, a former programmer at Informat. I take full responsibility for the present form of the system. The first version of Type & Set was installed in June 1988, but since then nearly every part has been rewritten.

What problems does Type & Set solve?

Using Type & Set rather than TEX incurs costs in running time and disk space. However, Type & Set solves or ameliorates the following problems, many of which are discussed in detail by Mittelbach [1]. The severity of these problems amply justifies the increased use of resources.

Page breaking is taken away from TEX completely and given to a program called PAGE which analyses the `DVI` file and writes a new `DVI` file, optimally paginated, with balanced columns, figure spaces, running material, headers and footers. PAGE takes its formatting information from a *style sheet* created using Type & Set's style sheet editor.

Varying numbers of columns. Type & Set can switch freely, as many times as you like (and as many times as you like *on the same page*), between text in one, two, three and four columns.

Baseline-to-baseline spacing occurs as a result of using PAGE rather than TEX to make up pages. All vertical dimensions in the Type & Set system are measured from the baseline of one line of text to the baseline of another. In particular, baselines at the bottom of pairs of columns align with each other, as do those of the last lines of text on facing pages. This also allows style sheets to specify a grid of lines on to which all baselines should fall if possible: that is, the y coordinate of a baseline should be an exact multiple of the grid interval.

Composite fonts (I prefer this term, suggested by Beebe [2], to the less descriptive 'virtual fonts') are used where necessary in the drivers. Readable data files called `FD` or 'FontData' files provide all the information a driver needs to convert a TEX character code into a device character, using transformations and superimposition if necessary. A utility, MAKETFM, is used to create `TFM` files for various output devices, given appropriate `FD` files and width tables.

Tables are created using a quasi-*wysiwyg* format in a text editor and converted into TEX by a program called TABLE. Horizontal spans, vertical and horizontal rules, and centring around any character (such as a decimal point), are all supported. Tables are very easily created and modified using this system.

Graphics is absent from TEX, and should not be added. The prevailing standard for graphics is PostScript, and so the Type & Set PostScript driver will pick up a named Encapsulated PostScript file, translate and scale it, and embed it in a figure space. The driver is told to do this by a `\special` written by a macro placed in the text and passed through by PAGE. A more general feature of PAGE, that can be used with any driver, is its ability to load, scale and embed a `DVI` file in exactly the same way.

Ease of use. Once a style sheet has been created for each kind of document to be typeset the rest is very easy. Staff at all levels of the publishing process, including those with no specialist computer knowledge (that is, nothing beyond basic abilities such as the use of the file system and rudimentary text editing) can be trained to use Type & Set in a day or two.

The rest of this article describes in detail how the problems were solved and how Type & Set works. What is described is a working system which was designed and implemented at a publishing house over a period of four years, and is now in daily use.

Data flow and general operation

Input. The user types a document using his or her favourite word processor. This may, for example, be Wordstar or WordPerfect, or (as I prefer, being in part an unreconstructed TEX hacker) an ordinary ASCII text editor. The document contains little or no TEX apart from markup codes known as *mode names* which are determined by the style sheet to be used. Mode names look like ordinary TEX control sequences, mainly because that is what they are. The preferred Type & Set style places mode names on separate lines. The mode determines all the stylistic and structural parameters of the text: its font, justification, indents, paragraph spacing, and whether it is part of the body text, a figure caption, or, say, a running header—and many other details. *Tag* is the term preferred in the world of desktop publishing, but we stay with *mode* for historic reasons.

Preprocessing. The first part of Type & Set to be run is the appropriate preprocessor for the text editor or word processor that has been used. In the case of Wordstar this is WS2TEX, which strips the high bits that Wordstar uses to mark the ends of words, converts Wordstar codes for italic, bold face, etc., into `\it`, `\bf`, etc., and emits standard ASCII text of the type TEX reads.

TEX. Any version of TEX can be used, with the proviso that if the document contains large tables a version with the biggest possible memory is desirable. TEX loads a customised format file, very similar to `plain.fmt` (indeed, *almost* upwardly compatible) called `tsplain.fmt`. The first command TEX finds in its input file is something like `\input mystyle.sty`, which loads the style sheet, which defines all the mode names and other markup codes used in the document. TEX then runs normally and writes a `DVI` file, using the minimal output routine from `tsplain.fmt`. This output file is effectively a *galley* in traditional typesetting terms, in that the text has been set in the desired fonts and *counted* or broken into lines, but has not yet been made up into pages. The `DVI` file contains numerous `\specials`, mostly for use by PAGE.

Page make-up. PAGE reads the `DVI` file and analyses it into lines, determining the mode of each line from a `\special`. A packet of information is built up for each line giving quick access to its mode, leading, position in the `DVI` file, and so on. At this point any *automatic material* is added. This consists mainly of the spacing and rules which the style sheet specifies for insertion before or after certain modes, or between paragraphs, or around blocks of text. PAGE then uses the line information to write a new file, given the extension `DVP`, but precisely conforming to the `DVI` format, which contains the made-up pages. If TEX fonts were used the work of Type & Set proper could end here, and the `DVP` file could be typeset using a third-party driver.

Previewing. Both `DVI` and `DVP` files for any printing device can be previewed on the screen using the Type & Set previewer, DVISCR, which draws characters using a device-independent vector font. This evades the problem of screen bitmap fonts not being available for, say, Optima on the Linotronic 100.

Proofing. Proofing is generally done on a laser printer, using the PostScript or Hewlett Packard LaserJet driver as appropriate. All drivers have the ability to emulate a font not found on the output device by using a similar one that *is* present; and the PostScript driver is especially optimised for emulation.

Printing. The driver family includes drivers for PostScript, Hewlett Packard LaserJet, Linotype devices using CORA V, Chelgraph devices using ACE, and Agfa Compugraphic devices. All drivers share common code which reads the `FD` (FontData) file, interprets the `DVI` file, and implements the composite font system.

Style sheets

The style sheet system both endows Type & Set with much of its power and limits it in various ways. Style sheets embody a generalisation about the possible forms of a document: a model which necessarily excludes some possible documents. The Type & Set style sheet model is designed to handle most types of journal and book design, but not magazines or newspapers, which in any case are laid out manually, page by page, rather than being intended for automatic page make-up.

Type & Set documents organise their text into two major divisions:

- body text, and
- running matter.

The body text is a single continuous sequence laid out over as many pages as necessary within a certain rectangle known as the *text area*, which may be positioned differently on left and right pages. Footnotes and figures are included within the broad heading of body text: these are positioned within the ordinary text under the control of *callouts* or references to them.

The running matter comprises running headers and footers, and folios (page numbers). These items

are placed in the margin outside the text area at fixed positions on each page. Style sheets allow you to specify different positions and different text for left, right, start and end pages.

Style sheets have three levels, *page*, *block* and *mode*. Each level has its own dialog within the interactive style sheet editor, STYLE. To create a style sheet a designer runs STYLE and fills in the boxes in the dialogs. The following paragraphs explain the meaning of each level.

Page level. This is where you specify the size and position of the page and of the text area within it. In the present version of Type & Set each document can have only a single page style: but multiple page styles are an obvious and not impossibly difficult extension which may be considered in the future.

Block level. You must create a block for each structurally different type of text in the document. Blocks are named objects belonging to one of the following *categories*, each of which has a two-letter symbolic name:

- `te` text
- `hs` running header, start page
- `he` running header, end page
- `hl` running header, left pages
- `hr` running header, right pages
- `fs` running footer, start page
- `fe` running footer, end page
- `fl` running footer, left pages
- `fr` running footer, right pages
- `fn` footnote
- `ps` folio, start page
- `pe` folio, end page
- `pl` folio, left pages
- `pr` folio, right pages
- `fi` figure

It is often necessary to have more than one text block. Blocks may be set in one, two, three or four columns: if you need to switch between text in different numbers of columns, as, for instance, in the case of a document with full-width single-column headings and two-column text, then the single-column text must have one block and the double-column text another.

The other main motivation for multiple text blocks is the need to position the blocks differently: each block may be offset from the left margin of the text area by a different amount, and this may be specified separately for left and right pages. This facility enables you to design a document (as in the case of a medical textbook published using Type & Set) where the headings project beyond the text, inward toward the margin.

Usually no more than one block will belong to each of the running header and running footer categories.

To sum up, the following information is specified at block level:

- name of the block
- category: see previous table
- offset from left margin
- absolute coordinates, unless category = text
- width
- number of columns
- gutter between columns
- weight of gutter rule, if any
- weight of box rule, if any
- margins inside box rule
- grid spacing, if any
- automatic spaces and rules
- explicit spaces, rules and indents

A typical simple style sheet will have six or seven blocks: a complex one will have twenty or thirty.

Mode level. This is the lowest level of description, corresponding to the *tags* used in desktop publishing packages such as Ventura Publisher. Here all the information about fonts and point sizes is stored, along with the justification and indents. The mode may be indented within its column, so the *measure* or, in TEX terms, the `\hsize` of a paragraph of text is determined by width of a column (set at block level) minus any left or right indents applied at mode level.

The name you give a mode is the actual markup code used in the input text, and may be any alphabetic sequence up to ten letters long. In the text all that is needed is a prefixed backslash: to invoke mode 'ref', the command `\ref` is used, on a line of its own.

Every mode belongs to a block, and normally several modes belong to the same block. In an extremely simple document consisting only of text and headings, there might be two modes, `\text`, in a fully justified roman font, and `\head`, in a left justified bold font. Automatic spacing can be used to insert space between the heading and the text.

Not one but four fonts are specified in every mode. These are roman, bold, italic, and bold italic, and will nearly always come from the same face or family unless special effects are intended. For example, a text mode might have Garamond Light, Garamond Book, Garamond Light Italic and Garamond Book Italic; within this mode the control sequences `\rm`, `\bf`, `\it` and `\bi` respectively would be used

to select each of the four fonts. Where the mode is inherently bold, as in a heading, the fonts are usually chosen so that roman and bold are identical, as are italic and bold italic.

Although Type & Set gives you control, via PAGE, over the degree of tolerance extended to widows and orphans, sometimes absolute prohibition of unwanted page and column breaks is preferred. This is done at mode level. For example, if you want the first two lines of each paragraph to be locked together and never split in any circumstances you can give *paragraph start lock* the value 2. Headings must never be separated from the text that follows, and this is done by specifying that the heading mode is to be locked to the next mode, as well as having all its lines locked together. Of course, this does not mean that all the heading lines in the document are locked into one huge block: the lock applies only to continuous sequences of lines belonging to the same mode.

To sum up, the following information is specified at mode level:

- name of the mode
- the block it belongs to
- the four fonts
- pointsize
- leading
- justification
- hyphenation tolerance
- looseness of word spacing
- left and right indents
- paragraph indent
- glue between paragraphs (`\parskip`)
- indent the first paragraph?
- lines to lock at start of mode
- lock all lines of mode together?
- lock this mode to following text?
- lines to lock at start of paragraph
- lines to lock at end of paragraph
- automatic spaces and rules
- explicit spaces, rules and indents

Table modes. I shall not deal with table modes in detail. They are at the same level of description as ordinary modes, and contain much of the same information, with the addition of some things needed specifically for tables, such as the amount of space to leave between the table and its caption, if any. Tables are explained below.

Fonts and font families

All text in a Type & Set document belongs to one of the modes of the style sheet in use. When STYLE creates a style sheet one of the files it writes is a large TEX macro package: each mode is a macro. This is how the fonts are selected. When a mode macro is interpreted by TEX, ten control sequences (among others) acquire new meanings:

- `\rm` roman text
- `\it` italics
- `\bf` bold face
- `\bi` bold italics
- `\sp` superscript
- `\sb` subscript
- `\mi` math italic font
- `\sy` symbol font
- `\mx` math extension font
- `\xx` Type & Set extension font

These invoke lower-level macros to select the appropriate fonts and pointsizes. The first six need not be used if WordStar or WordPerfect is used to input the text, because Type & Set can convert the control characters used by the word processors if necessary. The last four are also rarely seen in Type & Set text, but are invoked automatically in mathematical text and when special characters from those fonts are used.

The `xx` or Type & Set extension font is a ragbag of characters which seem to be required in journal and book publishing, and are generally provided on typesetting equipment, but are absent from the standard TEX layouts. These include solid triangles, solid circles, copyright symbols as single characters rather than composites, guillemets, etc.

Plain TEX preloads the sixteen most popular Computer Modern fonts and sets up a math font system at 10pt. The Type & Set format, `tsplain.fmt`, preloads no fonts at all: Type & Set is designed to be used on a wide variety of different devices, many with differing TFM files for fonts with the same names; so to preload fonts would cause confusion and errors.

The family mechanism used in plain TEX's math setting has to be retained, since it is hard-coded into TEX; but the other families are slightly different. Type & Set has:

family	*description*
0	text
1	math italic
2	symbol
3	math extension
4	italic
5	bold face
6	bold italic
7	Type & Set extension

No fonts are assigned to members of these families in `tsplain.fmt`. This is all done when the style sheet is loaded. For each mode, a font is assigned for all eight families at three different sizes, making a possible total of twenty-four fonts per mode. Very large style sheets may exhaust TEX's font memory, but in practice that does not happen very often, because many of the fonts belonging to one mode will be exactly the same as those of another; and STYLE is optimised to make use of any coincidences when writing the style sheet macros.

Superscripts and subscripts are implemented in a different way from plain TEX's method, except within math mode, where everything as far as possible is identical to plain TEX. Outside math mode the `\sp` and `\sb` macros provide more consistent text-mode superscripting and subscripting than _ and ^. They use TEX's font family system to determine the appropriate `\scriptfont` or `\scriptscriptfont` to use.

Type & Set's consistent approach allows everything to work in the same way whatever the current point size. In particular, mathematical setting is the same at any size, while remaining compatible with plain TEX.

Page make-up

`DVI` files written using Type & Set style sheets are completely standard and can be translated using any driver. All the extra information needed by PAGE, Type & Set's page make-up program, is to be found in *xxx* commands written by TEX's `\special` primitive.

PAGE is line-based: it analyses the `DVI` file into separate lines and moves these around, but goes no deeper than that except in the case of page numbers or *folios*, which must be often inserted into the middle of lines. Finding out where a line starts and ends in a `DVI` file originally seemed difficult, and a complex algorithm for finding minimal push-pop pairs enclosing pieces of text was used in an early version of PAGE, but eventually it was realised that with a little care one can ensure that every line is a first-level push-pop group.

PAGE can if needed optimise its layout over a whole document, by building a directed acyclic graph in which nodes are page breaks and arcs are possible pages labelled with their cost or 'badness', which is assessed using TEX-like criteria; and finding the lowest-cost traversal of the graph. In practice, however, users of Type & Set accept PAGE's first attempt at a solution, which is produced by successively taking the lowest cost for the current page and then moving on to consider the next. This willingness to accept compromise is caused by the slowness of PAGE when in whole-document optimisation mode, and obviously this is a shortcoming of the system. Nevertheless, people still find whole-document optimisation useful for improving documents which are badly laid out at the first attempt, usually because of problems with figure placement.

Blocks and column balancing. Pages are made up block by block. Each group of contiguous lines belonging to a single text block is collected together, with any figures called out somewhere among these lines, and any figures held over from previous pages. Lines are grouped into *shims*—bundles which cannot be split because they are locked together, or because they comprise a figure. (I have borrowed the term shim from Michael Plass [3, p. 36], who uses it in a slightly different way.) The list of shims for a block are then split into columns, and any mode-level space appearing at the top or bottom of a column is discarded. Block-level space is retained except when it appears at the top or bottom of a page.

Columns are balanced using a method that is similar, but not identical, to the method used by TEX for breaking paragraphs into lines. The method must be different, for the problem is different: paragraphs are split into an unknown number of lines, each of a known length, while in column balancing a known number of columns must be produced, each of an unknown length. The badness of a group of balanced columns is calculated in the same way that TEX uses for lines, using a function proportional to the cube of the glue ratio. Glue is set in such a way that the baselines of the bottom line of text in each column align together as well as those of the top lines. Since the height of the block is measured from its top baseline to its bottom baseline, this ensures that pages also align properly.

Figures. A figure in a Type & Set document is any section of the document starting with `\figure`, ending with `\endfigure`, and containing a sequence of figure spaces and captions. Figure spaces are inserted by writing `\figurespace <dimen>`, where `<dimen>` is the height of the space; and captions are chunks of arbitrary text in any mode belonging to a block of category `fi`.

PAGE extracts the figures and assigns each one a callout number which determines where it goes in the text. In fact, every contiguous sequence of lines of a certain category is given a callout number. This means that in a document consisting of some ordinary text, followed by a figure, followed by some more ordinary text, the callout numbers 0,

1, and 2 would be assigned to the three sections. This would cause the figure, callout number 1, to be placed somewhere after the first section of text. A figure's ideal position for Type & Set is where it appears in the original source text: the further away it ends up, the greater the penalty levied.

Internally figures are split into two groups: narrow and wide. Narrow figures span a single column in multi-column text, while wide figures are those which span all the columns of the block. (Type & Set cannot yet handle figures spanning more than one column but not all columns, such as two-column figures in a three-column block.) Wide figures are easy to place: they are inserted as soon as enough room is found, at or after their ideal position. Narrow figures are treated in a special way by the column-balancing system, in that they are allowed to float forwards from their ideal positions if the columns cannot be balanced otherwise. The algorithm which does this has recently been improved and now will very rarely fail to produce an acceptable page; but if there are just too many figures and not enough text, figures will inevitably appear one or more pages after their callouts.

PostScript and DVI embedding. Type & Set makes it very easy to embed PostScript pictures or existing DVI files in a document. The simplest way to do this is to place the command `\picture <filename>` on a line of its own in the source text. The `\picture` macro will write a `\special` to be read by PAGE, which then finds the file and determines its bounding box, deciding whether it is an encapsulated PostScript (EPS) or a DVI file from the first two bytes: `%!` for the former and bytes with the decimal values 247 and 2 for the latter. EPS files divulge their bounding boxes via a comment of the form `%%BoundingBox <lower-left-x> <lower-left-y> <upper-right-x> <upper-right-y>`, while for DVI files PAGE uses the values l and u from the postamble, unless it finds a `\special` of the form `page: bounds <n>,<n>,<n>,<n>`, which it interprets in the same way as the PostScript `BoundingBox` comment.

Having found out how big the figure is, PAGE scales it to fit the column width of the current mode. Unless the user has requested otherwise (via variants of `\picture` allowing greater control) the scaling is isomorphic: the width of the figure is scaled to be the same as the width of the column, then the height is adjusted to preserve the aspect ratio.

If the figure is a DVI file it is directly embedded in PAGE's output DVI file. To do this PAGE has to assign new numbers to the fonts in the embedded file so that they do not conflict with any in the main file; and all dimensions and fonts must be scaled by the appropriate amount as the file is read in. Only a single page, the first page of the file, is embedded: PAGE copies and scales everything between the BOP (beginning of page) and EOP, inserting a PUSH and a move to the correct x coordinate before the embedded code and a POP after it. The y coordinate need not be set explicitly: by the time PAGE has arrived at this point it will already be at the correct vertical position, since every line above the figure will have moved the current y coordinate down by its leading.

The procedure is different for PostScript files. PAGE calculates the coordinates and scaling factors needed and places them, with the filename, in a new `\special`, or rather, since it is not written by TEX, *xxx* command. DVIPS, the Type & Set PostScript driver, reads this information and loads the file in, creating a transformation matrix to scale and translate the embedded graphics.

Using this method one of Current Science's associated companies, Current Patents, publishes a journal giving details of the latest pharmacological patents. Diagrams of the chemical structures are created using commercial software which writes EPS files that are loaded with no modification into the PostScript output of Type & Set.

The following variants of `\picture` exist:

`\picture <filename>`: scale to fit column
`\apicture <filename>`: set at actual size
`\spicture <filename> <scale>`: set at given scale
`\wpicture <filename> <width>`: or given width
`\hpicture <filename> <height>`: or given height
`\xpicture <filename> <indent>`: or indented

A further variant, `\gpicture` or 'general picture,' gives you control over all the variables at once. In fact, all the above variants are expressed internally using `\gpicture`. Some examples:

```
\gpicture{mypic.ps}{2pc}{1in}{3cm}{0}{0}
  {0}
```

will be indented 2pc and forced to be 1in wide and 3cm high;

```
\gpicture{mypic.ps}{4pc}{20pc}{0pt}{0}
  {0}{0}
```

will be indented 4pc and forced to 20pc wide, and its height will be calculated from that: `0pt` means 'don't force'; and

```
\gpicture{mypic.ps}{0pt}{0pt}{0pt}{500}
  {750}{3}
```

will be scaled to half-size horizontally and three-quarters vertically and centred.

The `\gpicture` format is: `\gpicture {<file>} {<indent>} {<width>} {<height>} {<xscale>} {<yscale>} {<alignment>}`. If `<xscale>` is 0, `<width>` is used; and if `<yscale>` is 0, `<height>` is used. If `<alignment>` is non-zero and `<indent>` is zero the picture is aligned according to 1 = left, 2 = right, 3 = centre; otherwise it is aligned in the same way as the current mode. If `<width>` is zero it is calculated from height and vice versa. If `<width>` and `<height>` are zero both are calculated from the current column width.

Drivers and font layouts

The Type & Set drivers (with the exception of the screen previewer, which, being interactive, has to work in a different way) are all linked to two library packages, one to perform the basic `DVI` file interpretation, and the other to create data structures representing lines, phrases, words and characters, and to read the translation tables specifying the way TEX characters are rendered by device characters.

Thus most of a driver program is well-tested standard code, leaving only a small (300–400 lines of C code) device-dependent section containing the procedures needed to drive the actual printer or typesetter.

The `main()` function in a driver immediately calls the library function `DVImain()` with arguments giving the name of the default `FD` file to be read, whether accents are to be associated with the characters they are on or positioned separately, whether the device needs lines of text or separate characters, whether the output language is textual, like PostScript, or binary, like the Compugraphic language; and other information.

`DVImain()` has control for the entire run, calling other library functions to interpret the `FD` file and the `DVI` file and callback functions in the device-dependent module to set lines of text or individual characters.

Part of the reason for the simplicity of this approach is the use of a completely standard character layout on all devices. Type & Set is rigorously device-independent, like TEX itself, and apart from the font metrics no device-specific information is known at the time of running TEX or PAGE. Type & Set has a character set consisting of 640 characters, divided into five layouts:

- text
- math italic
- symbol
- math extension
- Type & Set extension

For a given printing device there are always *many* text fonts, but only *one* font in each of the other layouts: at least in logical terms, for the purposes of Type & Set. This reflects the fact that at sites using typesetting machinery such as the Linotron 100 and the Chelgraph IBX many text fonts exist, but only a few *pi* or symbol fonts. Similarly, PostScript provides a large number of text fonts but originally only one Symbol font.

The text layout is the same as that of plain TEX [4, p. 427] except for the following differences:

- character 14 changes from ffi to å
- character 15 changes from ffl to Å
- character 35 changes from # to £

These changes are all motivated by the need to have different variants of these characters in each text font, rather than have to use, say, the same pound Sterling with all the different fonts, whether bold or light, roman or italic.

The math italic, symbol and math extension layouts are identical to the plain TEX layouts [4, pp. 430–432]. This enables Type & Set to be 100% compatible with TEX mathematical setting, a feature which apart from its evident convenience absolves us from the task of writing a manual. The Type & Set extension layout, as described above, contains characters essential to book and journal publishing but entirely absent from the TEX layouts; and, since this is not the text font but one of the *pi* fonts, of which only one version exists per device, only symbols can go here, not textual characters. At present this font contains some forty characters. New accessions are made with reluctance, and only if the candidate character actually exists or can be emulated on most of the output devices.

As previously mentioned, drivers can elect to be passed complete lines or individual characters. There is flexibility too in the way a line is represented. A line-based driver will define a `DVIline()` callback function that is passed the address of a line structure. When a line arrives it is guaranteed to contain characters lying within a certain vertical distance of a common baseline, with no kerns greater than a certain size, and with no horizontal spaces greater than a predefined maximum. The idea is that drivers such as `DVICORA`, which generates Cora

V code for Linotron typesetters, can set the entire line at once, taking advantage of Cora's justification system and ability to interpret kerns, small amounts of up and down movement, and font changes within the line.

PostScript's `widthshow` primitive, however, although able to justify to a given measure, takes a string which must consist only of characters and spaces: movements and font changes have to be done separately. `DVIPS`, the PostScript driver, works at the *phrase* level rather than the *line* level. A line is a list of phrases, and phrases are defined in a much stricter way: each phrase is guaranteed to contain characters on precisely the same baseline, all in the same font, and possibly some spaces, each of which must be the same width within a certain predefined tolerance. This means that `DVIPS` can set each phrase using `widthshow`.

Each phrase is divided into *words*. These are composed of characters abutting horizontally, sharing a common font and baseline. Each character may have an associated accent. This last feature is used where accents are positioned by the device, and TEX's positioning must be discarded, as on Linotron devices using Cora V. No Type & Set drivers yet work at the word level, but the Compugraphic driver `DVICG` is an example of a character-level driver. These define a `DVIchar()` callback function that receives every character separately.

Composite fonts

A composite font is a font containing characters from more than one device font, or containing characters rendered by distorting or overlaying one or more device characters. Type & Set's composite font system is defined by `FD` or *FontData* files, one for each type of output device. An `FD` file is a readable ASCII file in a format modelled on that of Adobe Font Metric (`AFM`) files: that is, it is made up of sections starting with `Start<name>` and ending with `End<name>`, possibly nested, and within these sections there are data lines of the form `<key> <data>`. The main sections are `DevFonts`, mapping device font names or numbers to the names of their `AFM` files; `TSFonts`, mapping Type & Set fonts to device fonts; and several `Layout` sections, mapping standard Type & Set character codes to local character codes.

Rather than look at an exhaustive definition of the `FD` syntax and semantics, it will be more illuminating to follow two examples all the way from the `DVI` file to their representation in a typical output language, PostScript.

First, an ordinary character. Opcode number 12 is read from the `DVI` file and interpreted as 'set character 12 and move right by its escapement'. The current font is font 0, which has already been mapped to a `TFM` file called `time`. To convert the character into PostScript these two pieces of information are sufficient.

The `TFM` name `time` is is used as a key into a section in the `FD` file bracketed by `StartTSFonts` and `EndTSFonts`. This section associates Type & Set logical fonts with font layouts and names of `AFM` files, and contains the line `time`, meaning that, no layout having been specified, this font uses the default layout, given at the start of the `FD` file by the line `DefaultLayout text`. This tells the driver to look at the section starting with `StartLayout text` and ending with `EndLayout`. The character code, 12, is used as a key into a section within the layout called the `CharDefs` section and the line `12 174` is found, meaning that on this device Type & Set text character 12 (which, incidentally, is the fi ligature) is to be translated into character 174, the PostScript code for fi.

The PostScript font is determined by reference to a section starting `StartDevFonts`, which contains lines mapping the names of `AFM` or Adobe Font Metric files to a string of characters identifying the font on the device: in this case, the name `Times-Roman`. `AFM` files are used as the standard readable format for font metrics on all devices—not just for PostScript.

Now a composite character. This time the character code is 11, or the `ff` ligature, which does not exist in the PostScript text layout. If `TFM` files prepared specifically for PostScript fonts are used this character will of course never appear in the `DVI` file; but in this example we assume that `DVIPS` is being used for proofing, and must do its best to produce an emulation of something which will eventually appear, say, on the Chelgraph IBX typesetter, which *does* possess an ff ligature.

Everything happens in the same way until the layout line is reached, which is `11 102 # # 102 # [1 0 0 1 .25 0]`. Here `11` is the TEX character code, and the rest consists of two triplets, `102 # #` and `102 # [1 0 0 1 .25 0]`. Each triplet represents a device character, and is of the form `<code> <font> <transform>`, with `#` indicating that the default is to be used. The `<transform>` is a transformation matrix in the PostScript format [5, p. 65]. To render a composite character the output device takes each triplet in turn, setting the specified character from the specified device font, transforming it in the specified way. Here, if the second triplet had been identical to the first only a single f would have ap-

peared, since they would have been superimposed; but the transform on the second f moves it right by a quarter of an em, giving a tolerable rendition of an ff ligature.

You can see that the ordinary character in the first example uses the same syntax as the composite character once you know that any trailing # tokens can be dropped: the layout line for character 11 can be expressed more pedantically as `11 174 # #`.

At present there are two noticeable defects in the composite font system. The first is that a character cannot contain rules, and the second is that the transformation cannot make use of character metrics. It would be very useful to be able to define a transformation to move a character right by half the width of another character, but at the moment the x and y translation elements in a transform can be expressed only in ems, or, more accurately speaking, in fractions of the current pointsize.

FD files, as well as being read by the drivers, are used to generate TFM files for the various devices. For non-PostScript devices a utility called MAKEAFM creates AFM files from the width tables and other metrics supplied by manufacturers, while PostScript AFMs are downloaded via the Adobe PostScript Archive File Server. The AFMs are then used by another utility, MAKETFM, in conjunction with the FD files, to write the TFMs. This has to be done after any revision to the FD or AFM files that could affect character metrics.

Tables

Medical journals are full of tables, most of which contain spans, brace rules and numbers centred on a decimal point or a ± sign. The difficulty of creating tables in TEX makes an automatic table generator not only desirable but essential. The way Type & Set solves this problem is for the user to type the table in a sort of *wysiwyg* format, preferably, but not necessarily, using a special editor giving access to symbols representing column delimiters and rules. All the user has to do is ensure that the table is topologically equivalent to the desired result. Specifically, he or she must align the column breaks and arrange for the right things to span.

Each cell of the table can contain ordinary text, with or without TEX control sequences, plus special characters to tell the table processor how to align the cell. The absence of an alignment character causes the text in the cell to be centred. Macros are also available to insert multi-line paragraphs into cells if necessary. These are justified and hyphenated according to parameters defined in the table mode.

Here is an example: a modified and shortened version of a table in *The TEXbook* [4, p. 246]. The user creates a file called, say, `mytable.tab`, containing the following:

```
\narrow
|————————|————————————|
| Year   | World Population |
|————————|————————————|
| 8000 B.C. | 5,000,000       |
| 50 A.D.   | 200,000,000     |
|————————|————————————|
```

The vertical and horizontal rules here represent symbols taken from the standard IBM PC extended character set; but any symbols can be used, since the table system itself reads its special characters from a table. The control sequence `\narrow` is the table mode to be used.

When TABLE, the Type & Set table generator, is run, it translates the above into TEX code:

```
\vfil\eject
\narrow
\setbox0\hbox{a}
\boxtable{\offinterlineskip\tabskip=0pt
\halign to \hsize{\vrule # \tabskip=0pt%
&# \tabskip=\tg& # & #\tabskip=0pt%
&\vrule #&# \tabskip=\tg& # & #%
\tabskip=0pt&\vrule #\cr\vrzero height%
\hrzeroht&\multispan{3}\hrf{0}%
&\vrzero height\hrzeroht&\multispan{3}%
\hrf{0}&\vrzero height\hrzeroht\cr
\vrzero\global\tablepos=1&&\tsp Year%
\strut \tsp&&\vrzero%
&&\tsp World Population\tsp&&\vrzero\cr
\vrzero&\multispan{3}\hrf{0}&\vrzero&%
\multispan{3}\hrf{0}&\vrzero\cr
\vrzero&&\tsp 8000 B.C.\strut \tsp&&%
\vrzero&&\tsp 5,000,000\tsp&&\vrzero\cr
\vrzero&&\tsp 50 A.D.\strut \tsp&&\vrzero%
&&\tsp 200,000,000\tsp&&\vrzero\cr
\vrzero depth0pt\global\tablepos=2%
&\multispan{3}\hrf{0}&\vrzero depth0pt%
&\multispan{3}\hrf{0}&\vrzero depth0pt\cr
}}
\tablewrapup
\vfil\eject
\endinput
```

This contains many control sequences referring to parameters defined in the style sheet, such as the width of the table and the weight of the rules. A table is thus 'soft', in the sense that a given TAB file can be typeset in varying ways depending on the style sheet; or can even be used in two different

documents, looking different in each, because of differences in the definitions of the table mode. It is common practice to define a `\narrow` and a `\wide` table mode, and if a table exceeds the allowed width when typeset `\narrow`, it can be set using `\wide`. The table is included in the source text using the command `\input mytable`, and when PAGE analyses the DVI file it recognises the table as such and treats it as a figure, floating it to a convenient position. The code above produces the following result:

Year	World Population
8000 B.C.	5,000,000
50 A.D.	200,000,000

Conclusion

TEX is readily available, standard, stable, reliable and well documented. It is also complicated and hard to program in; and unsuitable for multi-column setting and baseline-to-baseline measurement. These facts have led us to use TEX's incomparable facilities for galley setting as the inner engine of our typesetting system, but to replace TEX's page make-up system with our own post-processor program, written not in TEX but in C. The TEX programming difficulties have been obviated by arranging for TEX macro packages to be written automatically by a style sheet editing program; and tables are coded not in TEX but on the screen in a *wysiwyg* format.

The other big TEX problem concerns fonts. Journals and books must be typeset using standard commercial faces such as Garamond, Optima, Helvetica, Univers, and Gill. TEX, while not in theory connected with any particular typeface or character set, is in practice closely bound to Knuth's Computer Modern family and its character set. The main achievement of Type & Set, apart from the page make-up system, is to retain almost complete compatibility with the plain TEX font layouts while typesetting on standard equipment—such as the Chelgraph IBX—using the standard typefaces available on the equipment.

These things make up Type & Set. This system has enabled us at Informat and Current Science to typeset about thirty academic journals and many other publications automatically using conventional typesetting equipment and standard fonts.

Afterword

Since this article was written two names have changed. The name of the software is to change from Type & Set to PageTEX, which, although similar to several other names of systems involving TEX, highlights the most important feature: automatic page make-up.

Informat Computer Communications is now subsumed into Life Science Communications, the holding organisation for Current Science and other companies, and it is Life Science Communications which will market PageTEX. Only the names have changed: the people and the software are the same.

References

1. Mittelbach, Frank. "E-TEX: Guidelines for Future TEX Extensions." *TUGboat*, 11(3): 337–345, September 1990.
2. Beebe, Nelson. Personal communication, September 1990.
3. Plass, Michael F. *Optimal pagination techniques for automatic typesetting systems*. Department of Computer Science, Stanford University, California, 1981.
4. Knuth, Donald E. *The TEXbook*. Addison-Wesley, May 1989.
5. Adobe Systems Incorporated. *PostScript Language Reference Manual*. Addison-Wesley, 1985.

◇ Graham Asher
Life Science Communications Ltd
34–42 Cleveland Street
London W1P 5FB
England
Telephone +44 81 348 1043

Philology

Computer Aided Hyphenation for Italian and Modern Latin

Claudio Beccari

Abstract

After an essential historical sketch of the evolution of latin into italian and modern latin, the peculiarities of both languages are described so as to understand the philosophy of the hyphenation patterns. The latter is one of the few cases where the same set is suitable for two different languages.

Sommario

Dopo aver delineato brevemente l'evoluzione del latino verso l'italiano e il latino moderno, vengono descritte le caratteristiche delle due lingue in modo da capire la filosofia dei *pattern* di divisione in sillabe. Questi *pattern* costituiscono uno dei pochi esempi applicabile a due lingue differenti.

Summarium

Latini sermonis evolutione ad italianum et latinum modernum breviter exposita, utrius sermonis specietates descriptae sunt ut philosophia de *pattern* ad syllabas dividendum intelligatur. Isti *pattern* duobus differentibus sermonibus applicabile exemplum sunt.

1 Outline of historical evolution

Classical latin, as we study it in schools and universities, is the language that was used, especially in written form, by the authors of the republican period and of the very beginning of the Roman empire. Common people used to speak a similar language that was open to the contribution of new words from other countries, to new constructs, and to a general simplification of the inflection of nouns, adjectives and verbs.

Cicero himself was complaining about the fact that common people (the *vulgus*) used to shorten the desinences leaving out the final consonants, and used to palatalize the 'c' and 'g' followed by the front vowels 'e' and 'i'. Those were the first signals of the autochthonous evolution of latin towards the modern language; in the other parts of the Roman empire similar evolutions were going on with a stronger influence of the native languages over which latin had superimposed itself; the invasions of the "barbarians" brought in peculiar pronunciations and a lot of lexical additions.

Latin decline was very slow because it was the scholar's, the chancellor's, the notary public's language for many centuries, and it was and still is the official language of the Roman Catholic Church; latin, in its modern form, is the official language of the Vatican State, and the daily Vatican newspaper, *L'Osservatore Romano*, is published mainly in italian, but with frequent contributions in latin, even commercial ads! Modern latin is used even for comic books; I suggest Snoopy [1], Mickey Mouse [2], Asterix [3][1].

Nowadays latin is studied in many countries as a regular subject in both high schools and universities; in Italy it is not classified as a "foreign" language and is a compulsory subject both in classical and scientific *licei* (high schools). In the past, latin was even more important in the education of young people; forty years ago I started latin in sixth grade and had eight years of it through junior high and high schools[2].

From the common people's language of the first century several regional and local dialects evolved; in 960 A.D. there is the first document explicitly written in what we might already call italian [4]; several documents, mostly poems, were produced in the following centuries, and by the end of the thirteenth century the masterpiece of Dante Alighieri, the *Divina Commedia*, is considered the main landmark of the new language, that was already so mature as to be used in a poetic treatise of history, philosophy and theology.

The modernization of Dante's language took place during the past seven centuries, but compared to modern italian there is not such a great difference as between the language used by Chaucer in his *Canterbury Tales* and modern english; today's Italian high school students can read Dante's poem and other even older texts with no more difficulty than that required by any other conceptual text.

[1] The first two books are intended as didactic aids for teaching latin, and are fully accented with both prosodic and rhythmic marks.

[2] I frequented the *liceo classico* and had also five years of classical greek; now I have an engineering degree and I am a professor of electric circuit theory. I am very glad I had the opportunity of completing my education by studying humanities for so long, and I wish the new generation could have the same.

2 Alphabet

Italian and modern latin use the 26 letter alphabet that everybody knows with the name of *latin alphabet*; actually there are some fine points to consider with due attention.

Italian. The letters J, K, X, Y, and W are used only in technical terms and symbols, foreign names, and some very specialized words, such as the international word *taxi*. J, K and Y survive in toponyms, family names, and english style nicknames, such as Stefy for Stefania (Stephanie). The letter J (see also below) used to be employed in the past as a graphic device to distinguish the semivowel role of the letter I, so that you have *Ajmone* (family name) and you may write *Iugoslavia* (modern spelling), *Jugoslavia* (old fashioned spelling), or *Yugoslavia* (international spelling) according to your preference; in italian all three are correct and are pronounced exactly the same way.

Besides the above mentioned letters, there are five vowels, none of which is mute: *a*, *e*, *i*, *o*, *u*; fifteen consonants: *b*, *c*, *d*, *f*, *g*, *l*, *m*, *n*, *p*, *q*, *r*, *s*, *t*, *v*, *z*; and one diacritical letter: *h*. The latter does not correspond to any sound and is used only to mark half a dozen words in order to distinguish them from similar ones that sound the same but have a different meaning, to mark some interjections, and to mark the velar pronunciation of 'c' and 'g' when otherwise they would be palatalized.

Except for a dozen articles, prepositions and adverbs (that nevertheless are used quite often), all common words in italian end with a vowel; of course this statement does not apply to trade marks, unassimilated foreign words, technical terms, and the like.

Another peculiarity is that every consonant may occur in its doubled form, and this corresponds to its reinforcement when the double consonant is pronounced. There are rare instances of double vowels, but in this case, contrary to what happens in english, they form different syllables instead of a diphthong; for example, *zoologico* can be divided as *zo-o-lo-gi-co*.

Latin. Classical latin missed J, U, and W, while V was used throughout wherever U or V are now used. Since the very beginning this anomaly was passed by scholars on into the spelling and printing of all languages; capital V was used in all circumstances, while 'v' was used in printing at the beginning of words and 'u' in the middle or at the end. This confusing habit was common to all western languages but fortunately it was abandoned starting in Holland during the sixteenth century; it lasted a little longer in Italy because of the wide use of latin, but was eventually done away by the end of the seventeenth century. When Knuth [5, p. 106] cites Pacioli's *Diuine Proportione*, published in Venice in 1509, he reports that title with the spelling of the original printing, but the pronunciation at that time already implied the consonant V instead of the vowel U.

In the middle ages and in the early times of printing it was the habit to use 'j' instead of 'i' in those cases where the letter 'i' formed a diphthong with the following vowel; it was just a graphic trick to distinguish the two roles of the letter 'i', and it was so successful that it was adopted also in other languages; this is the reason why even today we spell *junior* instead of *iunior*, although the latter is the formal latin spelling.

Modern latin uses both U and V in the proper positions, while J and W are used only in foreign names and toponyms.

There are six vowels: *a*, *e*, *i*, *o*, *u*, *y* and eighteen consonants: *b*, *c*, *d*, *f*, *g*, *h*, *k*, *l*, *m*, *n*, *p*, *q*, *r*, *s*, *t*, *v*, *x*, *z*. The ligatures *æ* and *œ* do not belong to latin; they were introduced in the sixteenth century in France and in England in order to replace the diphthongs *ae*, *oe*, and after that they enjoyed a certain popularity also in latin, but in modern usage, as well as in classical latin, these two diphthongs are spelled with separate letters.

3 Accents

Italian. In italian accents are used very sparingly; it is compulsory to mark with a suitable accent the last vowel of polysyllabic oxitone words (those that receive the stress on the last syllable), and to mark some well known and specified monosyllabic words that contain a diphthong. This is standardized by the Regulation UNI 6015 [7].

In contrast to spanish and portuguese, in italian there is no necessity to mark proparoxitone words with an accent, although the best grammars recommend doing so. In practice, if you exclude oxitone words (where accents are compulsory) and paroxitone words (where accents are not required), the other ones *may* be marked with an accent only when a different position of the stress might change the meaning; for example *làvati* means 'wash yourself' while *lavàti* is the masculine plural of 'washed'; in this circumstance it is advisable to mark the first case unless the meaning of the rest of the sentence does not make clear which case is implied. Although the 'Sommario' of this article contains five proparoxitone words, no accents were used.

The accent can be used also for denoting the open or closed nature of a vowel (only for tonic 'e' and 'o'), but this use is found only in dictionaries and grammars; a good grammar will certainly point out that *còlto* (picked up) is different from *cólto* (educated), but in practice the meaning is determined by the context while the actual pronunciation very strongly depends on the regional origin of the speaker.

The grave (`) accent is used on any vowel, while the acute (´) accent may be used only on the vowel 'e' (and on the vowel 'o', but only in optional situations) when it has a closed sound. Most Italians are not even aware of this choice; when they hand write, they just put any kind of small surd on the vowel to be accented, and by so doing they intend to mark only the stress; the tonic value of the accent is used only in dictionaries and grammars, while in printing the difference is maintained only for the letter 'e' in oxitone words more as a tribute to the tradition than for an actual semantic necessity.

When the accent is compulsory and upper case letters are used, if the character set does not contain accented vowels, it is accepted to use an apostrophe: UNITA' (unity) in place of UNITÀ; this practice is considered bad style in typesetting, but is used quite often in advertising.

The dieresis (¨) and the circumflex (ˆ) are not used anymore; in the past the dieresis was used in poetry to split a diphthong, and the circumflex had several meanings such as, for example, to mark the contraction of two 'i' into one sign in those plurals that centuries ago were spelled with a double 'i': *studii* (studies, two centuries ago), *studî* (one century ago), *studi* (modern).

Latin. In latin no accents are used; the breve (˘) and the long (¯) accents are used only in dictionaries, grammars and where prosody is dealt with. The dieresis is sometimes used in grammars and in prosody to mark the splitting of a diphthong: *aër* (air), *poëta* (poet).

4 Apocope and aphaeresis

Italian. In italian the dropping of one or more initial letters in a word (aphaeresis) takes place only in poetry and is marked with an apostrophe preceded by a white space.

The loss of one or more terminal letters in a word (apocope) either is not marked at all (see in the 'Sommario' *aver* in place of *avere*) or it is marked with an apostrophe when it corresponds to a vocalic elision before another vowel (see above *l'evoluzione* in place of *la evoluzione*) or to a complete syllabic apocope. The latter case is very unusual, while the vocalic elision is very frequent, so that this case must be taken care of properly when dealing with hyphenation; the rules stated in the Regulation UNI 6461 [6] require that when a line ends with an apostrophe, this *must not* be replaced again with the vowel it originally replaced. In the past, not too long ago, for example when I was in elementary school, the opposite rule was in use, so that there are occasional discussions between the old styled generation and the new one. Nevertheless even today it is considered bad style to end a line with an apostrophe, and in typography this practice is tolerated only when the line width is quite small, as in the daily newspapers' narrow columns.

Latin. I do not know of any case where apocope or aphaeresis are marked in any visible way; actually I am almost sure that these two spelling behaviours are not legal in latin.

5 Diphthongs

Italian. In italian a diphthong is formed by any vowel preceded or followed by an *unstressed* closed vowel ('i' or 'u'); so we have:

ia, ie, io, ai, ei, oi
ua, ue, uo, au, eu, ou
iu, ui

Italian diphthongs are always pronounced maintaining the sounds of the individual vowels, and the closed vowel plays the role of a semivowel or a glide.

There are also groups of three vowels that contain two semivowels or a semivowel and a glide:

iuo, uie
ieu, uoi, iei

An 'i' (possibly also a 'u', but I can't find examples) sandwiched between two other vowels behaves always as a semivowel, so it always starts a new syllable.

Latin. In latin there are more or less the same diphthongs as in italian with the addition of

ae, oe

that one or two centuries ago were replaced with the corresponding ligatures *æ*, *œ*; in modern latin the pronunciation of both these diphthongs is given by a single open 'e'[3].

[3] I have seen a reproduction of an italian book printed in Venice in the sixteen century where both these diphthongs were consistently replaced by their sound given by the letter 'e'.

Furthermore in some words of greek origin, latin may have the diphthong *yi*, for example *Harpyia* [9][4].

The main difference between italian and latin common diphthongs is that *ia*, *ie*, *io*, *iu* behave as such in latin only when they are at the beginning of a word or are preceded by another vowel; in any other case they are part of two different syllables; in italian they are always diphthongs provided the 'i' is unstressed.

6 Di- and trigraphs

Italian. In italian there are groups of two or three letters that imply a sound that is not implied by any other single letter of the alphabet; besides 'c' and 'g' modified with the diacritical 'h', and 'c' and 'g' modified with a diacritical 'i'[5] there are

gn, gli, sc

where *gn* is pronounced as in french, or as the spanish *ñ* or the portuguese *nh*; *sc* is pronounced as the english *sh* when it is followed by a front vowel 'e' or 'i', and *gli* is pronounced as the portuguese *lh* when it is not preceded by 'n' and is followed by another vowel or it is at the end of a word. These digraphs and trigraphs must not be split by the hyphenation process.

Latin. In latin by itself there are no indivisible digraphs or trigraphs, but since classical times the transliteration of greek words has required *th* in place of θ, *rh* in place of ρ (but *rrh* in place of $\rho\rho$), *ph* in place of ϕ, and *ch* in place of χ; therefore these digraphs cannot be split by the hyphenation process.

7 Hyphenation

Italian. The italian hyphenation rules are stated very simply as follows:

1. every syllable contains at least one vowel[6];
2. diphthongs and 'triphthongs' behave as one vowel;
3. a consonant followed by a vowel belongs to the same syllable as the vowel;
4. one or more consonants not followed by a vowel (at the end of a word, possibly because of an apocope, or in technical terms, trade marks and the like) belong to the same syllable as the preceding vowel;
5. when a group of consonants is found, the hyphen position is the *leftmost* one (even at the left of the whole group) such that the consonants that remain on the right of the hyphen can be found also at the beginning of an italian word;
6. prefixes and suffixes can be ignored and the compound word may be divided as if it were a single word; in any case division according to the etymology is accepted; in practice this happens only with the technical prefixes *dis-*, *post-*, *sub-*, *trans-*, which are not very common.

Once it is clear what is a consonant, a vowel, a diphthong and a 'triphthong', the only difficult rule to apply is the rule number 5; with the help of a school dictionary one can always find if there exists an italian word starting with a certain group of consonants.

The point is that if you use a dictionary of too high a quality, you will find words starting with almost any possible group of consonants: *bdelio*[7], *cnidio, ctenidio, ftalato, gmelinite, pneumatico, psicosi, pteridina, tmesi.* But many of these words, mostly of greek origin, do not find their way into school dictionaries (except *pneumatico* and *psicosi*), so that a diligent person will not be misled by too many technicalities and will find the correct division.

[4] One might think that it would be the same to consider the vowel 'y' and the diphthong 'ia', since the pronunciation would be practically the same; but if you look at it from the prosody point of view, the situation becomes completely reversed; a diphthong is always long while 'y' is always short, so that in prosody Har-pyi-a becomes $-\breve{}-$, while Har-py-ia becomes $-\breve{}\breve{}-$.

[5] In this case the letter 'i' does not form a diphthong with the following vowel but is used just to palatalize the two consonants; from the hyphenation point of view this subtle difference may be ignored.

[6] This rule applies to all languages, although in every language the notion of vowel is different; for example in several slavic languages 'r' is considered a vowel. If TEX contained a provision for this, the bad line break (compara-nds) that occurred in *TUGboat* 12, no. 2 (June 1991) on page 239, first column, 6–7 lines from the bottom, would not have taken place.

[7] Due to the extremely specialized nature of these words, I do not give the translation in english, because I did not find a suitable italian-english dictionary that reported them; I believe, though, that their scholarly nature is such that with minor modifications they exist also in english and many other languages.

b-d	b-n	b-s	c-m	c-n
c-s	c-t	c-z	d-g	d-m
d-v	f-t	g-m	p-n	p-s
p-t	p-z	t-m	t-n	z-t
g-fr	ld-m	ld-sp	l-st	mb-d
mp-s	nc-n	ng-st	n-scr	n-st
n-str	r-st	r-str	st-m	

Table 1: Groups of consonants that can be split across syllables

The Italian Standards Institute, in order to avoid confusion in this matter, established the Regulation UNI 6461 [6] that lists the groups of consonants that must be divided, table 1. This table does not list the normal consonant divisions, that is:

- digraphs and trigraphs can *never* be divided, except *gn* when it appears in a foreign word or in a word that derives from a foreign one and where the two letters are pronounced individually, such as *Wagner* → *wagneriano* → *wag-ne-ria-no*;
- geminated (double) consonants and *cq* must *always* be split;
- a liquid ('l', 'r') or a nasal ('m','n') is *always* separated from a following consonant except for the cases shown in table 1;
- any consonant is *never* separated from the following liquid except for the cases considered in the previous rule;
- the letter 's' is *never* separated from any following consonant (unless it is another 's').

The Regulation UNI 6461 states also the rules for the apostrophe, i.e. it behaves as the vowel it replaces; line breaking (without hyphen) is allowed after it when the line is very short, but it is bad style to do it[8].

Italian hyphenation for TEX has already been explained by Désarménien [8], but, although I wish I knew french as well as he knows italian, the 88 patterns that he created for italian were good only for consonants while completely ignoring diphthongs and 'triphthongs'; in a previous version I prepared for TEX 2.xx, 150 patterns were needed to perform italian hyphenation correctly.

For the rest the regulation is already made in such a way as to synthesize the hyphenation patterns TEX requires, without the need of running `patgen`; of course some care must be exercised in order to avoid strange situations and in order to replace TEX's inability to distinguish vowels from consonants.

With the advent of Version 3.xx of TEX it is better to set `\righthyphenmin` to the value 2, because there is no need to protect the hyphenation algorithm from the mute vowels ('e') that are so frequent in english; of course it is not good style to go to a new line with just two letters, but this is so rare that it is much better to give TEX more chances to find suitable line break points than to protect it from situations that in italian never take place.

Another reason for choosing this reduced value for `\righthyphenmin` is due to the accents; it was pointed out that in practice italian has accents, if any, only on the last ending vowel of a word. It is known that TEX does not hyphenate a word after an accent control sequence, but this drawback has a negligible influence on italian since after the accent control sequence the word may have just one letter; the accented letters found their way into the 256 symbol extended character set of TEX 3+ so that this simple drawback is eliminated, but even with the limitations of the 128 symbol character set (unless virtual fonts are used) this TEX peculiarity is of negligible influence.

With the reduced character set I admit that `virt\'u` → *virtù* (virtue) cannot be hyphenated because is too short, while with the extended set `virtù` could be hyphenated as *vir-tù*, but there are no problems with longer words, for example `qualit\'a` → *qualità* (quality) is hyphenated by TEX as *qualità*, the full possibility with the extended set being `qualità` → *qua-li-tà*. But with both character sets TEX gives correctly *per-ché* (because), *af-fin-ché* (so that), and so on.

There are no known problems with the synthesized patterns listed at the end; the only point that leaves me partially unsatisfied, but is grammatically perfectly correct, is the fact that technical prefixes such as *dis-*, *post-*, *sub-*, *trans-* must be explicitly separated with `\-` if one wants to stress their specific prefix nature. See below the solution for the same problem in latin.

Latin. The patterns that are listed at the end include a subset that was originally designed just for italian; with a little thought and a few additions the pattern set was extended so as to include also modern latin.

For what concerns diphthongs, italian and latin diphthongs were merged together under the assumption that TEX is not supposed to find every possible break point but only legal break points, so that if

[8] For line breaking after an apostrophe the new symbol `<cwm>` of the TEX 3+ extended character set may become useful.

two vowels are treated as a diphthong even if they belong to two different syllables, the only drawback is that you miss a legal break point but you do not make any wrong division. Moreover, most Italian readers feel uncomfortable when a break point is taken such that the new line starts with a vowel (this is certainly not the case with anglophone readers) so that the extension of the set of diphthongs of either language bothers neither Italian readers nor Latin ones. The declaration of æ and œ as letters with their `\lccode` allows also the hyphenation of words containing such ligatures.

Concerning consonant groups there is no regulation as for italian; my grammar [9] claims that latin hyphenation is done as in italian (except for what concerns prefixes and suffixes that must be divided etymologically) but in latin there are consonant groups that never occur in italian. Another book, [10], reports hyphenation rules for italian, classical latin, classical greek, french, german, english, and spanish; for latin the rules are drawn from a german source [11], which I was not able to reach, that apparently reports the hyphenation rules that were used in the middle ages. In classical times, as well as today, latin hyphenation is more similar to the italian than to what is reported in [10].

In order to find out how unusual consonant groups are treated in latin I examined an important scholar's book [12], the bilingual New Testament in greek and latin "apparato critico instructum", reprinted as a "reeditio photomechanica ex typographia ..., Romae" and for which "omnia iura reservantur"; clearly this is modern latin, although the book's contents, the latin part, contains the well known text that was translated from greek and aramaic by several authors across several centuries and copied by different copyists in many codices that are preserved all over the world. This critical edition is intended as a study material and is particularly cured in the language and the spelling for the very purpose of the book.

By examining the hyphenations of this book I could list a series of consonant groups, and I could realize that the digraph *gn* (which is such in italian but it is not supposed to be one in latin) was not treated in a uniform way, so as to have both *reg-num* and *re-gnum*. I decided to choose the second form of hyphenation for two reasons: a) it does not conflict with the italian rule, and b) the pronunciation recommended to the clergy and that is being used in the catholic universities, seminaries, monasteries, etc., corresponds to the italian digraph *gn*.

Also the letter 's' is not treated uniformly; it is generally treated as in italian, but there are cases where it is treated as in english; for example *blasphemia* (blasphemy) is hyphenated as *blas-phe-mia*. Since this does not conflict with the italian rule (in this language the group 'sph' is missing) a suitable pattern was generated in order to cope with such situations.

Some attention was given to prefixes and suffixes in order to find a way to separate them correctly according to their etymology; for what concerns prefixes, these must be separated regardless of the groups of letters that get split away, provided that the prefix did not lose its final vowel by elision with the initial vowel of the compound word's second element. For example the prefix *paene-* (almost) loses the last 'e' in *paeninsula* and therefore the whole word is treated as a single word and is hyphenated *pae-nin-su-la*.

It was possible to find suitable patterns for certain instances of *ab-*, *ad-*, *ob-*, *trans-*, for the prefixes *abs-*, *dis-*, *circum-*, *sub-*, and for the suffixes *-dem*, *-que* but the problem remains, although it shows up not so often.

The solution to this problem is to define a soft discretionary break that does not inhibit hyphenation in the rest of the word as does the plain definition `\-`. I chose to redefine the underscore character in such a way that in math mode it maintains the usual meaning of a subscript character, while in text mode it performs as a soft break.[9]

```
%
% New definition for the underscore
%
% Note that plain.tex and lplain.tex
% already \let\sb=_
%
\catcode`\@=11   % `at' is a letter
\catcode`\_=13   % Active underscore
\def\soft@break{\penalty\@M\hskip\z@\-
                \penalty\@M\hskip\z@}
\def_{\ifmmode \sb \else \soft@break \fi}
%
% With TeX 3+ use instead:
% \def_{\ifmmode \sb \else \cwm\cwm \fi}
%
```

Therefore, if wrong prefix or suffix hyphenations are found in the drafts, it is possible to correct (or to write it that way from the beginning) `con_iungo`, `ob_iurgo` so that the possible hyphenation points are *con-iun-go*, *ob-iur-go*.

[9] Editor's note: The active underscore must be treated with care in both TEX and LATEX; it cannot safely be used in `\labels` or in file names to be `\input` or `\included`.

8 Generation of the format file

In the appendix the file `italat.tex` is listed and the patterns may be checked against the rules that have been stated in the previous sections. Special attention was given to the groups *ps* and *pn*, because table 1 states that they must be separated, but the compound words with *psic-* (example *parapsicologia*) and *pneum-* (example *pseudopneumococco*) must not be hyphenated after the 'p'.

The ligatures 'æ' and 'œ' have been included with the `^^` notation and `\let` to the familiar control sequences `\ae` and `\oe` so that the pattern table is easily readable. If one has access to a TEX 3+ version, that allows the use of the 256 character code scheme published in *TUGboat*, [14], these ligatures have different codes so that suitable lines must be commented or uncommented.

The pattern list is preceded by some definitions:

- the category, lower case and upper case code definitions for the ligatures 'æ' and 'œ' so that they can be used in latin text; I stress again that these ligatures should not be used, except when quoting verbatim some text where they have been used.
- the new definition of the underscore character so as to produce a "soft" discretionary break; the TEX 3+ extended character codes described in [14], include the special invisible character `<cwm>` that can be used in the definition of such soft break, provided that the pattern table contains a suitable pattern.
- the pattern scheme was developed and tested with existing versions of TEX 3.0, 3.1, and 3.14[10], none of which accepts extended codes; if a version of TEX 3+ that accesses the extended character codes is available, the accented letters *à*, *è*, *é*, *ì*, *ò*, *ù* (and the corresponding uppercase letters) can be declared with their `\lccodes` by `initex`, so that the patterns require only simple control sequences in order to include such letters while remaining completely transportable.
- the definition of the new language "italian" with the command (`\italiano`) that invokes all the auxiliary definitions; the apostrophe character must be given its `\lccode` so as to treat it as a normal letter and as the vowel it replaces. Remembering that with the non-extended seven-bit ASCII and internal TEX codes, the apostrophe is used also as a single quote or as the first element of the ligature of the double quote, by treating the apostrophe as a letter, one might encounter rare instances where closing quote(s) introduce possible hyphen positions in the wrong places depending on the value of `\righthyphenmin`; every effort was spent to preview such cases, so that when apostrophes are present there are suitable patterns that allow or disallow hyphenation. Up to now no wrong hyphenations were reported in these cases.
- the command for latin (`\latino`, ablative and short for "latino sermone") is defined so as to catcode the æ and œ ligatures, and to restore the apostrophe to its original setting;
- the same is done with english (`\english`) so that you can interchange the three languages with the assurance of maintaining the correct settings for each one of them.

The patterns are enclosed within a group so that the `\lccode` of the apostrophe and the codes for the ligatures 'æ', 'œ', and accented vowels remain local and do not mix things up with the default language and/or with the previously defined languages.

Adding these hyphenation patterns to the format that has one or more languages already defined is not a heavy overhead; if you add italian and latin to the default language 'english' you do not need a large version of TEX; the statistics, after running `initex`, say that the hyphenation trie is of size 6359 with 223 ops, 181 of which are used for english and 42 for italian and latin; italo-latin hyphenation requires just 209 patterns (some of which probably never occur in practice) against the 4447 needed in english[11].

9 Conclusion

The hyphenation patterns valid for both italian and latin have been generated directly from the grammar hyphenation rules; for italian the set of patterns (a subset of that shown in the file `italat.tex` reported in the appendix) has been in use for two years in the Institution where I work, and after a short period of careful observation and debugging it performed absolutely without errors of any kind. Although the italian rules allow hyphenation of a compound word as if it were a simple one, some prefixes that are mainly used in technical terms may be explicitly hyphenated with the help of the new meaning of the underscore character.

[10] I have access to TurboTEX 3.0, to SBTEX 3.1 and to North Lake Software TEX 3.14a.

[11] These figures were obtained after an `initex` run with the `italat.tex` file in the appendix and no extended character set.

La lingua italiana e le lingue cosiddette romanze o neolatine, cioè lingue derivate anch'esse dal latino (francese, spagnolo, portoghese, rumeno ed altre minori), si fanno risalire all'*idioma*, che al tempo dell'impero romano era parlato nella penisola italiana, nelle regioni del Mediterraneo occidentale e nella Dacia, l'odierna Romania.

Tracce evidentissime si osservano ancor oggi non soltanto nel lessico e nella morfologia del gruppo linguistico neolatino, ma anche in altre lingue europee, quelle del gruppo anglo-sassone, come conseguenza dell'influsso diretto o indiretto esercitato dalla lingua di Roma sugli idiomi particolari dei popoli nordici.

Per quel che riguarda la lingua italiana, essa si collega direttamente al *sermo vulgaris latinus*, cioè al latino parlato comunemente dalle famiglie e in pubblico nei quotidiani rapporti di commercio e di affari.

Figure 1: Example of italian text typeset in narrow columns (from [9])

Et sicut Moyses exaltavit serpentem in deserto, ita exaltari oportet Filium hominis, ut omnis, qui credit in ipsum, non pereat, sed habeat vitam aeternam. Sic enim Deus dilexit mundum, ut Filium suum unigenitum daret, ut omnis qui credit in eum non pereat, sed habeat vitam aeternam. Non enim misit Deus Filium suum in mundum, ut iudicet mundum, sed ut salvetur mundus per ipsum. Qui credit in eum, non iudicatur; qui autem non credit, iam iudicatus est, quia non credit in nomine unigeniti Filii Dei. Hoc est autem iudicium, quia lux venit in mundum, et dilexerunt homines magis tenebras quam lucem; erant enim eorum mala opera. Omnis enim, qui male agit, odit lucem et non venit ad lucem, ut non arguantur opera eius; qui autem facit veritatem, venit ad lucem, ut manifestentur opera eius, quia in Deo sunt facta.

Figure 2: Example of latin text typeset in narrow columns (J 3,14–21)

For latin there is less experience but the impression is that also in this language there are no hyphenation errors; anyhow the author is grateful to anyone who might report suggestions and corrections. The new meaning of the underscore character is very useful for prefixes and suffixes and must be used whenever unusual consonant clusters are generated by the apposition of a prefix or a suffix.

In Figures 1 and 2 two examples show the performance of the hyphenation algorithm in italian and in latin when the line width is very small; you may notice than in such narrow columns italian gets some advantage thanks to the possibility of having two-letter final syllables.

I am pleased to express my thanks to the Nuns of the Benedictine Monastery of Viboldone in S. Giuliano Milanese (Milano), Italy, who assisted me with their experience in typesetting latin and other ancient languages.

References

[1] Schulz C.M., *Insuperabilis Snupius*, translated into latin by G. Angelino, European Language Institute, Recanati, Italy, 1984

[2] Walt Disney, *Michael Musculus et Regina Africae*, translated into latin by C. Egger, European Language Institute, Recanati, Italy, 1986

[3] Goshinny and Uderzo, *Asterix gladiator*, translated into latin by K.H.G. von Rothenburg (*Rubricastellanus*), Delta, Stuttgart, 1978

[4] Migliorini B.M., *Storia della lingua italiana*, (History of the italian language), Sansoni, Firenze 1963

[5] Graham R.L., Knuth D.E., Patashnik O., *Concrete mathematics*, Addison-Wesley Publ. Co., Reading, Mass., 1989 (3rd printing)

[6] *Divisione delle parole in fin di linea* (Word hyphenation at the end of a line), published by UNI, Ente Nazionale Italiano di Unificazione, Milano, 1969

[7] *Segnaccento obbligatorio nell'ortografia della lingua italiana* (Obligatory accent marks for the correct spelling of the italian language), published by UNI, Ente Nazionale Italiano di Unificazione, Milano, 1967

[8] Désarménien J., "The use of TEX in French: hyphenation and typography" in *TEX for scientific documentation*, D. Lucarella ed., Addison-Wesley Publ. Co., Reading, Mass., 1985

[9] Manna F., *Il latino ieri e oggi* (Latin yesterday and today), Signorelli, Milano, 1985

[10] Farina R., Marinone N., *Metodologia* (Methodology), Società Editrice Internazionale, Torino, 1979

[11] Leumann M., Hofmann J.B., Szantyr A., *Lateinische Grammatik* (Latin grammar), vol. I, München, 1977

[12] *Novum Testamentum Graece et Latine* (The New Testament In Greek and Latin), A. Merk S.J. ed., Istituto Biblico Pontificio, Roma, 1984

[13] Braams J., *Babel, a multilingual style option system for use with LaTeX's standard document styles*, *TUGboat* 12, no. 2, (June 1991), pp. 291–301

[14] Ferguson M.J., *Report on multilingual activities*, *TUGboat* 11, no. 4 (November 1990), pp. 514–516

◇ Claudio Beccari
Dipartimento di Elettronica
Politecnico di Torino
I-10129 Turin, Italy
beccari@polito.it

Appendix A The `italat.tex` file

This file must be input after the last line of the file `plain.tex` (or `lplain.tex` for LaTeX); the definitions given before the pattern table are better located in the format file, so they are valid for any style and there is no possibility to forget them. If this file is used with TeX 2.xx, comment out the lines that contain the commands `\newlanguage` and `\language`. Similarly, move the comment character as indicated if you use this file with TeX 3+ and the extended character set.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%                       F I L E    I T A L A T . T E X
%
%                  Hyphenation patterns for Italian and Latin
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%          Prepared by Claudio Beccari, Politecnico di Torino, Italy
%                           e-mail beccari@polito.it
%
% Version date  10 January 1992
%
\catcode`\@=11                                          % @ is a letter
% Useful definitions
%
% Ligatures \ae, \AE, \oe and \OE
%
% If TeX 2.xx is used, or no extended character set is available,
% the macro \specialcodes is correct;  comment it if you use TeX 3+
%
%%%%% \ae = 26  \AE = 29  \oe = 27  \OE = 30
%
\def\specialcodes{%
          \catcode 26=11 \catcode 29=11 \lccode 29=26    % Ligatures \ae,\AE
          \uccode  29=29 \lccode 26=26  \uccode 26=29
          \catcode 27=11 \catcode 30=11 \lccode 30=27    % Ligatures \oe,\OE
          \uccode  30=30 \lccode 27=27  \uccode 27=30}
%
% If TeX 3+ is used, and assuming that lccodes and uccodes are already
% established, the following \specialcodes macro must be uncommented:
%
%%%%    \`a=^^e0   \`e=^^e8   \'e=^^e9   \`\i=^^ec   \`o=^^f2   \`u=^^f9
%%%%    \`A=^^c0   \`E=^^c8   \'E=^^c9   \`I=^^cc   \`O=^^d2   \`U=^^d9
%
%\def\@namelet#1{\expandafter\let\csname#1\endcsname=}
%\def\specialcodes{\@namelet{@gr@a}^^e0   \@namelet{@gr@A}^^c0
%                  \@namelet{@gr@e}^^e8   \@namelet{@gr@E}^^c8
%                  \@namelet{@ac@e}^^e9   \@namelet{@ac@E}^^c9
```

```
%                  \@namelet{@gr@i}^^ec   \@namelet{@gr@I}^^cc
%                  \@namelet{@gr@o}^^f2   \@namelet{@gr@O}^^d2
%                  \@namelet{@gr@u}^^f9   \@namelet{@gr@U}^^d9
%                  \let\cwm=^^17 \lccode`\^^17="17}    % Compound word marker
%
% Languages
%
% A number is given to italian/latin hyphenation
%
\newlanguage\italian
%
% New definition for the underscore
%
% Note that plain.tex and lplain.tex  already \let\sb=_
%
\catcode`\_=13                                          % Active underscore
\def\soft@break{\penalty\@M\hskip\z@\-\penalty\@M\hskip\z@}
\def_{\ifmmode \sb \else \soft@break \fi}
%
% With TeX 3+ use instead:
% \def_{\ifmmode \sb \else \cwm\cwm \fi}
%
% Definition of the commands \italiano and \latino
%
\def\italiano{\language=\italian \specialcodes
              \righthyphenmin=2 \lccode`\'=`\'}
%
\def\latino{\language=\italian \specialcodes
              \righthyphenmin=2 \lccode`\'=0}
%
\def\english{\language=0  \righthyphenmin=3  \lccode`\'=0} % Needed in order
%                 to restore the english settings when you revert to english
%
{                                                     % Beginning of the group
%
% With TeX version 3 or lower use:
\let\ae=^^1a \let\oe=^^1b                               % Comment with TeX 3+
% and with TeX 3+ use
% \let\ae=^^e6 \let\oe=^^f7                            % Uncomment with TeX 3+
% and
% \def\`#1{\csname @gr@#1\endcsname}                   % Uncomment with TeX 3+
% \def\'#1{\csname @ac@#1\endcsname}                   % Uncomment with TeX 3+
%
\language\italian \specialcodes
\lccode`\'=`\'                              % Apostrophe catcoded to lower case
\patterns{
.a2b2s3  .a2b3l
.o2b3l   .o2b3m .o2b3r .o2b3s
.an1ti3      .an1ti3m4n
.di2s3ci3ne  .cir1cu2m3
.wa2g3n      .a2p3n .ca4p5s
.pre3i       .pro3i
.ri3a .ri3e  .re3i  .ri3o  .ri3u
.su4b3lu     .su4b3r
2s3que.      2s3dem.
3p4si3c4     3p4neu1
a1a   a2e    a2i    a2j    a1o   a2u  a2y             % Diphtongs and hiati
a2y3o a3i2a  a3i2e  a3i2o  a3i2u ae3u
e1a   e1e    e2i    e2j    e2o   e2u  e2y   e3iu
```

```
i2a   i2e    i1i    i2o    i2u   io3i
o1a   o2e    o2i    o2j    o1o   o2u  o2y
o3i2a o3i2e  o3i2o  o3i2u
u2a   u2e    u2i    u2o    u1u   uo3u u3i2a u3i2o
\ae1  \oe1
%i2\'a0 i2\'e0 i2\'e0 i2\'o0 i2\'u0                  % Uncomment with TeX 3+
%u2\'a0 u2\'e0 u2\'e0 u2\'i0 u2\'o0                  % Uncomment with TeX 3+
1b2   2b3b   4b3d   2b3n   2b3t                     % Consonant groups
      2b3s4a 2b3s4e 2b3s4i 2b3s4o 2b3s4u  2b3s4t   u2b3s4c
1c2   2c3c   2c3m   2c3n   2c3q  2c3s  2c3t  2c3z  2ch3h
1d2   2d3d   2d3g   2d3m   2d3s  2d3v  4d3w
1f2   2f3f   2f3t
1g2   2g3g   2g3d   2g3f   2g3m  2g3s
1h2   1j2    2j3j   1k2    2k3k
1l2a  1l2e   1l2i   1l2j   1l2o 1l2u   1l2y
      2l3l   l3f4t  2l4l3m 1l'  1l\ae1 1l2\oe1
%1l2\'a0 1l2\'e0 1l2\'e0 1l2\'i0 1l2\'o0 1l2\'u0     % Uncomment with TeX 3+
1m2   2m3m   2m3b   2m3p   2m3l 2m3n  2m3r   2m4p3s
      2m4p3t 4m3w
1n2a  1n2e   1n2i   1n2j   1n2o 1n2u  1n2y   2n3n   n2c1n  2nil
      n2g3n  2n1r   n2s3m  n2s3f 2n'  1n2\ae1        1n2\oe1
%1n2\'a0 1n2\'e0 1n2\'e0 1n2\'i0 1n2\'o0 1n2\'u0     % Uncomment with TeX 3+
1p2   2p3p   2p3s   2p3n   2p3t 2p3z  2ph3p  2ph3t  2s3p2h
1q2   2q3q
1r2a  1r2e   1r2i   1r2j   1r2o 1r2u  1r2y   1r2h   1r2\ae1 1r2\oe1
%1r2\'a0 1r2\'e0 1r2\'e0 1r2\'i0 1r2\'o0 1r2\'u0     % Uncomment with TeX 3+
1s2   2s3s   2st3m  2s'
1t2   2t3t   4t3m   2t3n   1t'  4t3w
1v2   2v3v   1w2   2w3w   wa4r
1x2a  1x2e   1x2i   1x2o   1x2u 1x2y  2x3x  1x2\ae1 1x2\oe1
%1x2\'a0 1x2\'e0 1x2\'e0 1x2\'i0 1x2\'o0 1x2\'u0     % Uncomment with TeX 3+
y2a   y2e    y2i    y2o    y2u
1z2   2z3z   2z3t   1z'
%\cwm1\cwm0                                         % Uncomment with TeX 3+
}
}                                                        % End of the group
% At this point it might be necessary to restore the at sign catcode
% \catcode`\@=12                                    % Uncomment if necessary
```

Fonts

Invisibility using virtual fonts

Sebastian Rahtz

Abstract

The SLiTEX 'invisible' fonts are currently produced by a special set of METAFONT files; an alternative method of generating 'invisible' versions of any font is presented, using virtual fonts.

1 Introduction

As soon as Donald Knuth announced [Knuth 1990] that 'virtual fonts' would be an official part of the TEX version 3 distribution, I started to think of ways in which they could make life easier for me. Now that drivers exist which support the new font format (I have used Tom Rokicki's `dvips` for PostScript output, and Eberhard Mattes' driver family distributed with emTEX for screen previewing), I have been able to realize some of these ideas. In a forthcoming article for the journal of the UK TEX Users Group, *Baskerville*, I discuss the use of virtual fonts in a PostScript environment to implement the suggested extended font layout for TEX; here I look at a very simple use of virtual fonts to generate the invisible fonts needed by SLiTEX.

SLiTEX needs a set of fonts which produce no output, but use up the same amount of white space as the main fonts, in order to produce overlays [Lamport 1986, pp. 136–37]. Leslie Lamport achieved the desired effect by taking the set of fonts he was using for SLiTEX anyway, and altering the METAFONT source so that they produced no marks on the paper but had the same metrics as the parent font. Despite continued complaints that these fonts are not found in all LATEX distributions, the system works well, provided that one sticks to the default fonts. The big disadvantage, of course, is that if one uses a different typeface one has to go right back to METAFONT sources (which may be unavailable or non-existent) to generate invisibility. But who ever dared change the font setup in SLiTEX anyway? The exciting work of Mittelbach and Schöpf [Mittelbach & Schöpf 1990] changes the situation, however. It is now very easy to build a SLiTEX which uses a different set of fonts; but if we decide to do our slides in, say, Optima, how do we get invisibility? There are three approaches:

1. If we are using a PostScript printer, there is a simple, and elegant, solution. Where we want invisible text, we simply typeset it in the parent font, but bracket it with a pair of `\special` commands which instruct PostScript to set these letters in white, i.e., invisible.[1] This has the great advantage that only one set of font metrics is needed for TEX, but has the disadvantage that it is dependent on the printing device. Leslie Lamport (*pers. comm.*) has written an appropriate style file for SLiTEX which does the necessary work (this is *not* part of the LATEX distribution at present).
2. We could tinker slightly with TEX to do the same thing, i.e., read a single font metric file, but in a certain mode produce only white space rather than the characters. So far as I am aware, this has not been done. Whether it is possible by arcane TEX macros, I feel incompetent to judge! The helpful reviewer of this paper suggests, as a start:

   ```
   \catcode`a = \active
   \defa{\setbox0=\hbox{a}\hskip\wd0\relax}
   ```

   Chris Torek's 'mctex' distribution goes in this direction by allowing for a mapping file read by the drivers which lets you assign the names of conventional `tfm` files to the invisible names, and add an attribute of invisibility; this forces the driver to look up the width in the `tfm` file, and move right by the required amount.
3. We can take an intermediate route, and generate a set of font metrics which satisfy TEX itself about the size of characters in the font, but actually point to virtual fonts which produce just white space. This is the approach I have adopted. Its advantage is that it is portable, and applicable to any font, but has the disadvantage that TEX still has to load two sets of font metrics, which are identical.

2 Methodology

Even if we have no METAFONT source for a font, we certainly have a TEX `tfm` file. We therefore start with that as our basis, and extract from it the widths of every character. This data can be used to write a virtual font which has an entry for every character, but whose body simply contains a `dvi` instruction to move sideways by the width of the original character.

As the format `tfm` is not very easy to read, let us look at the work that needs to be done in the

[1] This scheme would fail, not if we printed on coloured paper, but if we overlaid text on some other colour, in which case PostScript's imaging model would produce white letters.

human-readable pl form. The format of the vpl file, the corresponding human-readable form of the vf file, is a superset of the pl format [Knuth 1990, p. 18ff.]. The first part of the file, which describes the basic characteristics of the font, can be left more or less unchanged, adding just a VTITLE line; for example:

```
(VTITLE created on Sunday, October 28, 1990
  11:28 pm)
(COMMENT an invisible form of the font)
(FAMILY TIMES-ROMAN)
(CODINGSCHEME ADOBESTANDARDENCODING)
(DESIGNSIZE R 10.0)
(COMMENT DESIGNSIZE IS IN POINTS)
(COMMENT OTHER SIZES ARE MULTIPLES OF DESIGNSIZE)
(FONTDIMEN
   (SLANT R 0.0)
   (SPACE R 0.25)
   (STRETCH R 0.3)
   (SHRINK R 0.1)
   (XHEIGHT R 0.448)
   (QUAD R 1.0)
   )
```

A typical entry for a character in the pl file looks like this:

```
(CHARACTER C a
   (CHARWD R 0.444)
   (CHARHT R 0.458)
   (CHARDP R 0.014)
   )
```

so all we need do is reproduce this, and duplicate the width in a MOVERIGHT command:

```
(CHARACTER C a
   (CHARWD R 0.444)
   (CHARHT R 0.458)
   (CHARDP R 0.014)
   (MAP
    (MOVERIGHT R 0.444))
)
```

There is no need to worry about heights and depths, as the dvi file will contain commands to reposition the current print position at the start of each line, and TEX will have ensured that the baseline separation is correct. Even characters like floating accents, such as the circumflex from Times-Roman, have a negative 'depth' to tell TEX that they occupy only the top area of the box:

```
(CHARACTER O 303
   (CHARWD R 0.333)
   (CHARHT R 0.675)
   (CHARDP R -0.494999)
   )
```

This presents no difficulties because it is TEX which has done all the work establishing where the accent is to go, and has written instructions into the dvi file to position us there. The simple MOVERIGHT should continue, therefore, to work.

3 Implementation

An ideal conversion program would read a tfm file and write a vf file. I have adopted a simpler approach, which is to write a utility in the text-processing language Icon which reads an ASCII pl file and writes a vpl file. My program consists of the following steps, illustrating how easily a pl can be parsed:

1. run tftopl on the base font
2. open the pl file
3. read and concatenate lines until matching sets of (and) are found; process result.
4. for each element that is a CHARACTER, copy out the CHARWD value as a MAP ... MOVERIGHT command at the end of the entry.
5. simply copy out other elements (e.g. FONTDIMEN).
6. call vptovf.

When the resulting vf and tfm files are installed, we are ready for action. If we started with a file called ptmr.tfm, we generate ptmrj.tfm and ptmrj.vf[2]. TEX is told the invisible font is called ptmrj, and reads the metric file. The dvi driver finds ptmrj.vf, obeys the MOVERIGHT instructions, and no further characters are typeset.

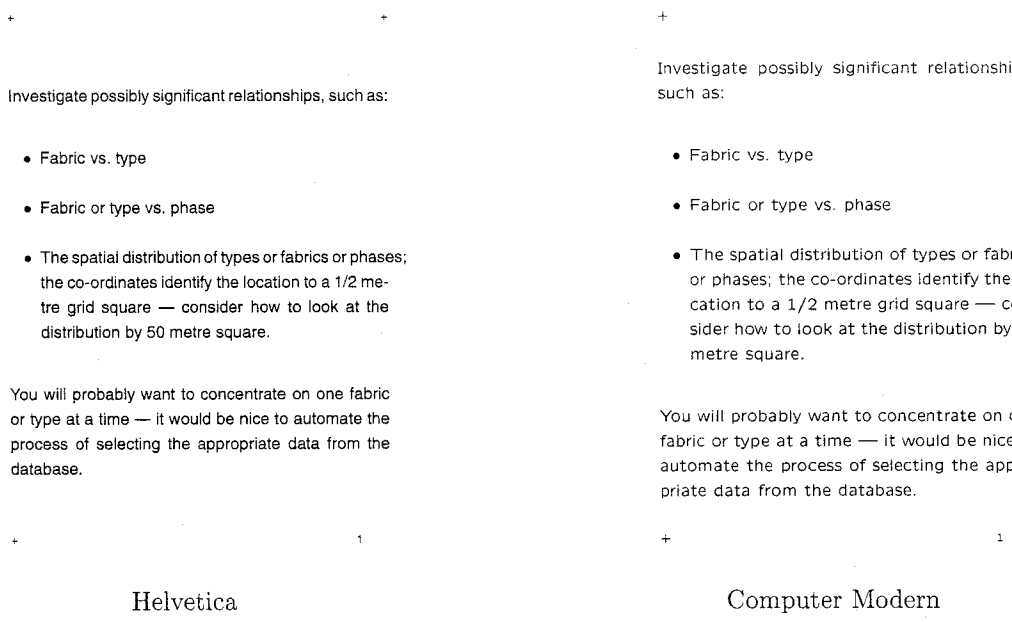
4 Results and acknowledgements

This idea has not been fully tested, and TEX may have tricks up its sleeve to waylay us. But it does work in simple examples (including things like accentuation), and I have some confidence that a portable tftoiv program can be written to make this a general technique. Those who use a PostScript printer will find the 'write-white' approach more attractive (especially as it saves on font space) and extensible, but less portable.

Russell Lang (rjl@au.edu.monash.cc.monu1) has translated my rough Icon program into C, and this latter version is available from him or me for interested parties who care to rewrite the whole thing in WEB for the TEX community, or make it read and write tfm and vf files.

Frank Mittelbach has written an experimental redefinition of the \invisible macro in SLiTEX which takes the ingenious route of simply prepending 'iv' to the name of the current font family, and then calling \selectfont to pull out the correct font, bypassing the hard-wired font names in the

[2] Using Berry's font naming scheme, and adding a 'j' attribute for invisibility



Helvetica Computer Modern

Figure 1: The effect of various fonts in slides

original SLiTEX. Future releases of the font selection macros are likely to feature this system; combined with the virtual font suggestions outlined above, one can envisage a long-overdue renaissance for SLiTEX.

To demonstrate the effect of the virtual font approach in these pages would be difficult, but it may be of interest to readers to see the visual effect (albeit reduced) of SLiTEX slides set in Helvetica rather than the familiar hugely expanded Computer Modern (Fig. 1).

This article has benefited considerably from comments by the *TUGboat* reviewer.

References

[Knuth 1990] KNUTH, D. 1990. 'Virtual fonts: more fun for grand wizards', *TUGboat* 11, no. 1, pp. 13–23.

[Lamport 1986] LAMPORT, L. 1986. *LaTeX User's Guide & Reference Manual*, Addison-Wesley Publishing Inc., Reading, Massachusetts.

[Mittelbach & Schöpf 1990] MITTELBACH, F. AND R. SCHÖPF 1990. 'The new font family selection—User interface to standard LaTeX', *TUGboat* 11, no. 1, pp. 91–97.

◇ Sebastian Rahtz
ArchaeoInformatica
5 Granary Court
St Andrewgate
York Y01 2JR
U.K.

Packing METAFONTs into POSTSCRIPT

Toby Thain

Aimed at implementors of DVI-to-POSTSCRIPT translators, this article suggests adapting Rokicki's packed font format [1] to compactly define bitmap fonts in POSTSCRIPT, an approach which has been successfully implemented and tested by the author.

The problem of integrating METAFONT and POSTSCRIPT has been tackled in two completely different ways: by modifying METAFONT to output curvilinear paths and outlines [4, 5], and by using METAFONT's standard bitmap output directly. Since POSTSCRIPT allows flexibility in representation, the choice is largely philosophical. While outlines are less device-dependent and more amenable to linear transformations, this author feels that TEX users need an effective means of using METAFONT-generated bitmaps with the gamut of POSTSCRIPT devices.

Another consideration is that METAFONT's digitisation is likely to be better than that produced from a machine-translated outline font; current POSTSCRIPT printers are notably lax in this regard. (Adobe Type Manager is a significant improvement, but printers do not yet incorporate this renderer, resulting in the irony that some non-POSTSCRIPT printers using ATM render text better than many POSTSCRIPT printers.) In short, where low-resolution devices are concerned, the author believes that METAFONTs such as Computer Modern

Name	dpi	GF	PK	PS	VM
cmr10	300	13 040	5352	8370	14 948
"	600	24 124	10 808	13 919	20 404
"	1270	49 468	27 576	30 983	37 134
"	2540	108 828	59 492	62 424	68 360
cmsy10	2540	121 548	73 992	76 951	82 840
pkdict				2091	6846

Roman are better digitised by METAFONT than by POSTSCRIPT from an outline. (This is also apparent from examples presented in [5].)

It is clear from the table that GF-format glyphs are excessively large at high resolutions, and even low-resolution fonts are cumbersome if no compression is used. Another constraint is the limited 'virtual memory' available to POSTSCRIPT devices; all fonts in a document typically coexist in this space, and therefore it is important to manage it efficiently (the column headed 'VM' indicates how much of this memory is used by each packed font.) A compressed representation is also desirable where fonts are stored in a POSTSCRIPT device's local file system.

This wastefulness suggests using a compressed description which could be interpreted by the printer itself. Having the advantages of high compression ratios and well-documented algorithms, PK format seemed a natural choice. Furthermore, existing TEX fonts are largely in PK format.

An implementation of this idea is shown in Figure 1, encapsulated in a dictionary named pkdict. Most of the work is done by the bc procedure; what remains is to show how specific fonts are arranged to make use of that code. (These examples, compromises between clarity and brevity, can be improved in either direction.)

An example

A bitmap font dictionary might be defined as follows (automatically generated from logo10.300pk):

```
/logo10 pkdict begin 10 dict dup begin
/FontType 3 def
/FontMatrix [26214400 109226469 div
   0 0 2 index 0 0] def
/Encoding ev def
/BuildChar /bc load def
/glyphs 128 array
77 [27 25 -3 24 33 0 t 12
   <f3d87d69fd4bfd29f13d031623b32624a326339336347346
   437346f535356f6333667313767776858 6f9396fd830> p
69 [20 25 -3 24 26 0 t 11 <dfeac5fc62e73c5dc0> p
84 [23 25 0 24 23 0 t 10 <d3ebba3a> p
65 [22 25 -3 24 28 0 f 12
   <5c8d3558543c33f3d131e63d3025e9d330> p
70 [20 25 -3 24 26 0 t 11 <dfeac5fc62e93c50> p
79 [26 25 -1 24 28 0 f 12
   <7cbd5768654d1443d333f3d531eb3d73f13d5324d3436a65
   d79d16> p
78 [22 25 -3 24 28 0 t 12
   <3d37d28d19d0ac613c623b624a6349644865476646674568
   4469436a426b326c316cad09d18d27e2d330> p
def
/FontBBox [0 0 30 25] def
end end definefont
```

```
/pkdict 11 dict dup begin /a [0 128 192 224 240 248 252 254] def
/p { ] 2 index 3 1 roll put} bind def /f /false load def /t /true load def
/r {string currentfile exch readstring pop} bind def /ev [0 1 255 {( ) dup 0 4 -1 roll put cvn} for] def
/gn {d p get hi {-4 bitshift /hi false def} {15 and /p p 1 add def /hi true def} ifelse} bind def
/gb {d p get i and 0 eq /i i dup 1 eq {/p p 1 add def pop 128} {-1 bitshift} ifelse def} bind def
/pb {{j or} if j 1 eq {1 q 3 -1 roll put /q q 1 add def 0 128} {j -1 bitshift} ifelse /j exch def} bind def
/pn { gn dup 0 eq { {1 add gn dup 0 ne {exit} {pop} ifelse} loop
    exch {4 bitshift gn or} repeat 15 sub 13 df sub 4 bitshift df add add
  } { dup df gt { dup 14 lt {df sub 1 sub 4 bitshift gn add df add 1 add}
    {14 eq {pn} {1} ifelse /r exch def pn} ifelse } if } ifelse } bind def
/bc { save 3 1 roll exch /glyphs get exch get pkdict begin 16 dict begin aload pop
  /d exch def /df exch def /c exch def /w 6 index def
  3 index neg 3 index 6 index sub 1 add w 6 index sub 5 index 1 add setcachedevice
  false [1 0 0 -1 8 -2 roll 1 add] /p 0 def
  w 7 add -3 bitshift dup string /l exch def dup string /br exch def
  dup string exch 1 sub 0 1 3 -1 roll {1 index exch 255 put} for /wr exch def
  df 14 eq {/i 128 def {/j 128 def /q 0 def 0 w {gb pb} repeat j 128 ne {l q 3 -1 roll put} {pop} ifelse l}}
  { /r -1 def /hi true def /n 0 def
    { r 0 lt { /r 0 def /k 0 def /v 0 def /q 0 def /b w def /n n { dup 0 eq {pop pn /c c not def} if
      dup b lt dup {1 index} {b} ifelse dup
      k 0 ne { dup 8 k sub 2 copy gt {exch} if pop c {dup a exch get k neg bitshift v or /v exch def} if
        dup k add dup 8 eq {l q v put /q q 1 add def pop 0} if /k exch def sub } if
      dup 0 ne { dup -3 bitshift dup 0 ne { l q c {wr} {br} ifelse 0 4 index getinterval putinterval
        q add /q exch def 7 and } {pop} ifelse c {dup a exch get} {0} ifelse /v exch def k add /k exch def
      } {pop} ifelse exch {dup b exch sub /b exch def sub} {k 0 ne {l q v put} if sub exit} ifelse
    } loop def } if /r r 1 sub def l } } ifelse imagemask end end restore } bind def
end def
```

Figure 1. It looks complicated, but it works!

Such Type 3 fonts require a `BuildChar` procedure to draw glyphs. The `Encoding` array trivially maps the 256 possible character codes to 256 different names (required by the font cache). The `FontMatrix` defines the resolution; the font's pixels and measurements such as bounding boxes and widths are transformed by this matrix to 'user space'. `glyphs` is an array mapping character codes to 'packets' describing the character:

[w h h_off v_off dx dy f dyn_f data].

f is `true` if the first run of pixels is black, `false` otherwise, and *data* consists of the nybble-packed glyph description. The other entries correspond to PK character packet fields.

After the character descriptions comes the font's bounding box, which is computed by the PK-to-PostScript translator.

Finally, the `BuildChar` procedure (which takes its definition from `bc` in `pkdict` above) is executed by the interpreter when a character from this font is needed. In short, `bc` looks up the character's description given the font dictionary and a character code, passes the width and bounding box to `set\-cache\-device` (thereby requesting that the character be cached), and draws the glyph one row at a time using `imagemask`. For further information on the structure of user-defined PostScript fonts, see [2].

PostScript's font cache ensures that each different glyph drawn is only decompressed once; subsequent requests for the same glyph bypass `BuildChar` and are satisfied by the cache. Only glyphs which are drawn are decompressed.

It is worth noting that some device-independence comes 'for free' with PostScript; for example, bitmap fonts defined in this manner can be rendered at any resolution, under any linear transformation; hence, if bitmaps are not available for a specific resolution or magnification, those at another resolution can be substituted (albeit with less pleasing results).

Character positioning is an issue which can be dealt with on two levels. METAFONT records two different dimensions for each character: a rounded 'displacement' in pixels, and a 'TFM width' for typesetting purposes. Two considerations conflict when positioning characters: it is desirable that characters are spaced according to their TFM width, so that margins line up; and yet integral displacements will yield more even spacing (an issue addressed in more detail by [6]). The approach taken by this PostScript representation is that characters are spaced by displacement, therefore the DVI-to-PostScript translator will occasionally need to compensate for accumulated rounding errors according to [6].

If fonts are to be stored on a printer's local disk (freeing the host of the responsibility of managing them), a file `usr/pkdict` should be created containing the definition of `pkdict` above. Each font file should begin with

```
/pkdict where {pop} {(usr/pkdict) run} ifelse
```

which executes this file if necessary, rather than defining the dictionary in each font file. Font file names are conventionally prefaced by '`fonts/`' (e.g., `fonts/logo10`). Local font files are executed by `findfont` to define the font dictionary, and therefore consist of the same text that would be downloaded 'on the fly' (see [3]).

Finally, the author would welcome discussion of the techniques and issues presented in this article, and any aspects of the integration of TeX and PostScript. The author has also ported TeX 3·0, METAFONT 2·0 and associated tools to the Inmos T800 Transputer.

References

1. T. Rokicki, "Packed (PK) font file format," *TUGboat* **6**(3), pp. 115–120 (1985).
2. *POSTSCRIPT Language Reference Manual,* Adobe Systems Inc., Addison-Wesley (1985).
3. *POSTSCRIPT Language Program Design,* Adobe Systems Inc., Addison-Wesley (1988).
4. J.D. Hobby, "A METAFONT-like System with PostScript Output," *TUGboat* **10**(4), pp. 505–512 (1989).
5. S. Yanai & D.M. Berry, "Environment for Translating METAFONT to PostScript," *TUGboat* **11**(4), pp. 525–541 (1990).
6. D.E. Knuth, `DVItype.web` (1982–90).

◇ Toby Thain
RMB 712
Beaufort
Victoria 3373
Australia
Ph. +61 53 497296
Fax +61 53 497339

Complete Greek with Adjunct Fonts

C. Mylonas and R. Whitney

We present a set of good quality Greek fonts and a system for accessing the full Greek alphabet with all diacritic marks using 'adjunct' fonts without explicit font changing commands, and a system of Greek monitor characters permitting direct TEXing.

Introduction. The project started with an innocuous attempt by one of the authors (it is obvious who) to learn TEX for translating a Greek book on Mechanics, only to find that TEX lacked the bold upright Greek lowercase letters used for symbolizing vectors such as $\boldsymbol{\omega}$ and $\dot{\boldsymbol{\varphi}}$ for angular velocity vectors. After some trials with variations of "poor man's bold" (*The TEXbook*, p.386) it became clear that the only hope of getting good characters was to produce them with METAFONT. The beautiful pictures of *Computer Modern Typefaces* whetted the desire for complete fonts for writing Greek, especially as TEX's Greek mathematical symbols were not intended for writing Greek text (*The TEXbook*, p.430), and furthermore, they have to be entered with lengthy commands. At that time the ever helpful Barbara Beeton brought to our attention the excellent, original and efficacious Greek system and fonts of Professor Silvio Levy [1], which he graciously offered us. That is when the collaboration of the two authors started, the one ever entangled in problems resolved by the other. At first all efforts were concentrated on the design of the Greek characters. The problem of limited font capacity was postponed till the end. In this report, however, it seemed more useful to start with the requirements for a Greek font, which we then set to meet. Accordingly the following paragraphs describe the characters required in a complete Greek font and the constraints imposed by TEX; the chosen system for accommodating all requirements and the development of the method of adjunct fonts; the commands for typesetting Greek; the design of a new family of Greek fonts; and the use of Greek monitor characters.

Greek fonts within TEX's constraints. The biggest problem in font making for TEX is the limited number of characters in each font. TEX originally would only accept direct input of 128 symbols in each font, but this was increased to 256 in the new version TEX 3 of 1990. Even this larger capacity is insufficient for all characters and symbols of a complete Greek font, much more the smaller font size still in wide use. We shall show how two or more 128-position fonts can be interconnected so as to furnish most of the Greek characters, and how the same system could be applied to two 256-position fonts, which would then accommodate all Greek characters with room to spare for symbols useful in lexicography or instruction.

Classical Greek uses most of the first 295 symbols shown in Table I, and lexicography some or all of the last 65, making about 360 (without counting the new characters on line 4 or some of the symbols on block 6). The six accented vowels marked with a '−' sign in blocks 2 and 3 do not appear in Classical Greek, but classicists want them for indicating the pronunciation in ancient dialects. The characters in parentheses with asterisk at the end of blocks 2 and 7, and their uppercase counterparts in lines 13 and 9 seem to occur very rarely, if at all, in Classical Greek literature [2]. All these symbols do not fit even in a 256-position font. Furthermore, though over 64000 ligatures and kerns can be used in TEX 3 and can be provided by METAFONT 2.7, only 256 are allowed in METAFONT 1.7 (we use PCMF v.1.7 [11]), so that we had to count and be sparing in their use. If all diacritized characters (i.e., characters with accent, breathing or iota subscript) were to be accessed as ligatures so as to permit automatic hyphenation, about 200 ligatures would be required for Classical Greek and many more for lexicography. Some 55 kerns would then be left for Classical Greek, whereas our Greek alphabet is less regular and requires more kerning than the Latin.

Modern Greek is written with a single accent and diaeresis but no breathings or iota-subscript (blocks 1 and 5 in Table I, and 15 characters of block 2). This reduces the number of characters to 103 (with the punctuation marks and numerals) and of ligatures to 23, so that a 128-position font is sufficient, with 233 kerns available. Even this number of kerns is not high: in the worst case every lowercase letter would need kerning with itself and every other, which with the 7 simply accented vowels, makes $24 \times 32 = 768$ kerns (no letter after a final sigma). The possible kerns between vowels with breathings (always first in a word) and the other unaccented letters is $56 \times 25 = 1400$. Another $24^2 = 576$ would be required for kerning uppercase letters, and 768 for uppercase followed by lowercase, bringing the maximum total to about 3500 kerns. Happily, many character pairs need no kerning and, of course, all

TABLE I
A COMPLETE GREEK FONT

			Clas.	Mod.
1.	` α ο κ ν ζ η θ ι κ λ μ ν ξ ο π ρ σ ς τ υ φ χ ψ ω	25	25
2.	´ ` ~ ά ὰ ᾶ έ ὲ ε῀ – ή ὴ ῆ ί ὶ ῖ ό ὸ ο῀ – ύ ὺ ῦ ώ ὼ ῶ ῝ ῎ ·· ΐ ῒ ϊ ΰ ˙ ϋ (῝ ῏ ῦ ΰ)*	66 33	- 15
3.	᾽ ᾽ ᾽ ῾ ῾ ῾ ῾ ῾ ᾰ ᾱ ᾲ ᾳ ᾴ ᾶ ᾷ ᾱ ᾰ ἐ ἑ ἒ ἓ ἔ ἕ Ἐ ἑ – ᾐ ᾑ ᾒ ᾓ ᾔ ᾕ ᾖ ᾗ ἰ ἱ ἲ ἳ ἴ ἵ ἶ ἷ ὀ ὁ ὂ ὃ ὄ ὅ Ὸ ὁ – ὐ ὑ ὒ ὓ ὔ ὕ ὖ ὗ ᾠ ᾡ ᾢ ᾣ ᾤ ᾥ ᾦ ᾧ ῤ ῥ	66	-
4.	ϐ ϐ́ ϐ̀ ϐ̃ ϐ̔ ϐ̓ ϐ̈ ϐ̄ ϐ́ ϐ̋ ϐ̆ ϐ̌ ϴ	-	(13)
5.	A B Γ Δ E Z H Θ I K Λ M N Ξ O Π P Σ T Y Φ X Ψ Ω Ϊ Ϋ	26	26
6.	, . · : / ; ! ’ + = @ % * [ ] ( ) ϐ « » " - – — ¯ ˘ §	22	27
7.	ᾳ ᾴ ᾲ ᾷ ᾲ ᾴ ᾷ ᾳ ᾴ ᾷ (ᾲ ᾴ)* ῃ ῄ ῂ ῇ ῂ ῄ ῇ ῃ ῄ ῇ (ῂ ῄ) ῳ ῴ ῲ ῷ ῲ ῴ ῷ ῳ ῴ ῷ (ῲ ῴ)	31	-
8.	A E H I O Y Ω with accent-breathing on the left	56	-
9.	Ἀ Ἡ Ὠ plain or with accent-breathing on the left*	21	-
10.	Digits	10	10
11.	Digamma, Greek numerals, left subscript: Ϝ ϛ ϟ ϡ ,	5	-
	Total	295	103

Lexicographic or other use:

12.	Ἰ Ὑ with accent-breathing on the right (beginning diphthongs)	16	-
13.	A E H I O Y Ω Ἀ Ἡ Ὠ Ἰ Ὑ with accent on the right*	33	-
14.	Upper and lowercase vowels with brève (˘) or macron (¯)	16	-
	Total	65	
	Grand total	360	

pair combinations do not normally occur, though the transliterated outlandish names seen today in the press contain surprising combinations. In the system described below we had to be content with 225 kerns, and we ignored the rarer and the less evident misspacings, eagerly awaiting the full implementation of TEX 3 and METAFONT 2.7 with the promised practically unlimited kerning.

No concern about kerning is indicated by Professor Levy who introduces an ingenious scheme for printing the character 's' as final sigma 'ς' when at the end of a word and as middle sigma 'σ' otherwise, but uses 56 ligatures of the middle sigma 'σ' with every other letter and accented vowel. This dispensation from choosing the proper sigma is a luxury not enjoyed by any typist in Greece, but would be welcome if different keys weren't anyway needed for the two sigmas in word processors producing Greek characters on the screen (see section *Greek Monitor Characters*). Another 128-position distribution was proposed by Haralambous and Thull [4] in their excellent compromise for Modern Greek, and a general 256-position distribution by Haralambous [5] was communicated to us after the submission of the present article.

The chosen system. It was felt that a "complete" and unique Greek font should be produced for typesetting both Modern and Classical Greek without font-changing commands and that it should include all the additional accents and pronunciation signs used in lexicography. In the absence of a large font size with unlimited ligatures and kerns, a system with the following characteristics was chosen:

a. The number of distinct characters in the Classical font was reduced to 187, without sacrificing its capabilities, by retaining only the diacritized vowels of blocks 2 and 3 in Table I, and abandoning the separate characters representing the vowels with iota subscript (block 7 and line 9), the uppercase vowels with accent- breathing on the left (line 8), and the diacritized vowels for lexicography (lines 12–14). These diacritics are instead entered correctly as separate characters, the accent-breathing before uppercase vowels and the iota subscript after vowels.

b. The iota subscript was made a separate character of zero width and a left offset which centers it under the plain or accented vowels α η ω when typed after them. The subscript is placed at the right of the uppercase Α Η Ω with the macro \I which offsets it by .4em (or with just "|" when with breathing). Subscripted vowels are mostly first or last letters in a word, and will run no risk of hyphenation if separation after the first or before the last letter of a word is excluded. Internal subscript vowels are rare, and even more rarely will hyphenation occur at their subscript, but this should be checked. Likewise the accent-breathing to the left of uppercase vowels run no risk of hyphenation.

c. In lexicography, only main entry words or the word following them are diacritized with *brève* and *macron* or, when rarely capitalized, with accent or accent-breathing on the right or left. But main entries in a dictionary are always placed at the beginning of a line and need no hyphenation. The accent-breathings of Table I, line 12, and the accents of line 13 occur in rare uppercase words in lexicography or instruction and are entered on the right of the letter. The brève and macron of line 14 are entered with plain TEX commands (*The TEXbook*, p.52, 356), e.g. "`\u o`" gives "ŏ".

d. The 103 Modern Greek characters double as Classical when the single accent is marked as a regular 'acute', as is done in most books in Greece [3], but not when it is shown as a novel triangular mark suggested by some Greek grammarians. The 192 independent Classical characters are then reduced to 89, of which 25 can be placed with the 103 characters of Modern Greek in one 128-position *main Greek font.* The question then remained how to sort out at least 64 classical characters into a separate 128-position *classical adjunct font*, and how to access them easily, preferably without explicit font-changing commands.

The solution was found by noting, first, that the 66 diacritized characters of Table I block 3 have the common trait of carrying a smooth or rough breathing, hence are entered with a leading symbol '>' or '<' followed by an optional accent and finally a vowel or 'ρ' with which they form ligatures, and second, that if these 'breathed' characters were placed in the classical adjunct font, the common breathing symbols could be made to act as commands to shift and access the compound character in that font, without any additional entry or any hint to the user that a second font is used. Adjunct fonts match the main fonts in style and magnification, and the shift is automatic to the matching adjunct font. The two fonts double the possible number of ligatures and may be easily joined into one (without any changes in the .tex files) when 256-position systems become available with sufficient kerns and ligatures. It would even be possible to fit most of the uppercase vowels with breathings in the adjunct font so as to access them by ligature, but this was not found

TABLE II
PRESENT KEY CORRESPONDENCE

α	β	γ	δ	ε	ζ	η	θ	ι	κ	λ	μ	ν	ξ	ο	π	ρ	σ	ς	τ	υ	φ	χ	ψ	ω	ϑ
a	b	g	d	e	z	h		i	k	l	m	n		o	p	r		s	t	u	f		y		
							⇓						⇓				⇓					⇓		⇓	⇓
Not obvious							j						x				c					q		w	v

Α	Β	Γ	Δ	Ε	Ζ	Η	Θ	Ι	Κ	Λ	Μ	Ν	Ξ	Ο	Π	Ρ	ϐ	Σ	Τ	Υ	Φ	Χ	Ψ	Ω	ϑ
A	B	G	D	E	Z	H		I	K	L	M	N		O	P	R		S	T	U	F		Y		
							⇓						⇓				⇓					⇓		⇓	⇓
Not obvious							J						X				C					Q		W	V

′	‵	~	’	”	”‵	῀	‘	“	“‵	῁	¨	΅	῭											
´	`	~	>	>´	>`	>~	<	<´	<`	<~	¨	¨,	¨‘	¦										
,	.	:	/	;	·	!	’	-	–	—	+	=	@	%										
,	.	:	/	?	;	!	’’	-	--	---	+	=	@	%										
[	]	(	)	«	»	″	§	ϝ	ς	ϛ	ϡ	‘												
[	]	(	)	((	))	""	\pg	\f	\s	\n	\p	\x												

necessary as the separate accent-breathings worked perfectly. The required inputs for writing Greek are shown in section *Typesetting Greek* and on lines 7 and 8 of Table II. The only drawback of this system is the exclusion of kerning between characters of different fonts. But the adjunct font contains the vowels with breathing, which come always first in a word and may need kerning only on their right. This may frequently be circumvented by carefully adjusting the character margins (`adjust_fit`).

Accordingly two interrelated 128-position fonts were used: the main font containing all characters, symbols, and punctuation marks used in Modern Greek, as well as the remaining simply accented vowels from blocks 1,2,4,5,6,10 of Table I (plus two new characters), and the adjunct font with the remaining vowels with breathing from blocks 2 and 4, plus some additional characters. The main font `gm10` and its adjunct `gp10` are shown in Table IV. Numerals, plain alphabets, simple accents and all punctuation marks have the same positions as in Levy and the corresponding characters in Computer Modern. The remaining characters in `gm10` and all of `gp10` are in unrelated positions.

The correspondence between Latin and Greek characters, shown in Table II, and the way of entering the accents and breathings, are almost identical to Levy's, though we use 128-position fonts. Minor differences are the use of separate keys for middle and final sigma, and the introduction of some new characters. The key correspondence is mostly obvious: the non-obvious are specifically marked in Table II. S.A. Fulling [6] places only two characters in positions different from Levy's and ours, and suggests working for a consensus, and Y. Haralambous [5] is forming a special group for standardizing Greek TEX. Such a standardization is most desirable, but it would require agreement with other major groups of users even of plain Greek, as it would be odd to use different keyboard conventions in TEX- and nonTEX-Greek. Such groups are the classicists, who use some other variations, and the mass of users in Greece, whose support will be essential for a successful adoption, and who mostly use the IBM mapping with Θ Ξ X Ω Ψ respectively on keys U J X V C.

Typesetting Greek. The macros used in Greek mode are gathered in the file `greekmac.tex`. To typeset a document in Greek one has only to enter `\input greekmac` at the start of the file and then the brief command `\[` to start Greek mode in font

gm10 (see *The Fonts* below), and finally \] to end it. When clashing with other TeX extensions, e.g. with LaTeX , these commands may be redefined to \(...\), at a loss only of the command \(in the concert format (*The TeXbook,* p.409).

These commands replace Levy's longer \begingreek and \endgreek (which can also be used), as well as his alternative $ for starting and ending the Greek mode after the command \greekdelims which substitutes \math for the math-mode command $. Our \[and \] leave $ untouched, they resemble delimiters, and have proved quite convenient during frequent alternations between Greek and English, especially when intermixed with math. All TeX control sequences are valid in Greek mode, e.g., \rm switches to cmr10 and \mz back to gm10 (but for macros defined in Greek mode see Levy [1]).

Things get even simpler and TeXing faster if one creates a new format file greek.fmt containing greekmac.tex, and dispenses with \input greekmac in the text file (though \[will still be needed to start Greek). To TeX one then enters

```
tex &greek <file>
```

In Greek mode one can write Modern Greek using ondy the acute accent, or Classical Greek with all diacritics, and can shift to other Greek or English fonts as shown in the following example of a Greek .tex file. Greek mode must be closed at the end of the file to avoid the TeX complaint \end occurred inside a group at level 1.

```
\[  >Arq‘h <ellhniko~u keim’enou.
\rm Or in Modern uniaccent Greek:
\mz Arq’h ellhniko’u keim’enou.
\rm Which means:  \it Beginning of
Greek text.  \]
```

This gives

Ἀρχὴ ἑλληνικοῦ κειμένου.
Or in Modern uniaccent Greek:
Αρχή ελληνικού κειμένου.
Which means: *Beginning of Greek text.*

Accents and breathings are entered as shown in the 1st and 3rd of the next four lines, and produce the 2nd and 4th lines respectively:

```
’a  ‘i  ~h  >e  >’a| >‘h| >~a  <h  <‘o  <~w|
```
ά ὶ ῆ ἐ ᾄ ᾒ ἆ ἡ ὃ ᾧ

```
A’  I‘  H~  >E  >’A\I  H\I  >~A  <H  <‘O  <~W|
```
Ά Ὶ Η͂ Ἐ ᾌ ῌ Ἆ Ἡ Ὃ ᾯ

Typewriter mode *within* Greek mode should be enclosed in curly brackets: {\tt ...} as it causes a redefinition of the catcodes of >, >, |, which must revert to their greek-mode values. Mathmode works perfectly within Greek mode: the symbols >, <, | automatically regain their mathematical meaning and Greek fonts can be specified within mathmode

$\mbz a\cdotp b<|a||b|$ *gives:* $\boldsymbol{\alpha}\cdot\boldsymbol{\beta} < |\boldsymbol{\alpha}||\boldsymbol{\beta}|$

Some Rare Occurrences. Although efforts were made to cover most of the requirements of Greek typesetting, some rare cases must be made up with improvised commands, as e.g., the rare characters shown at the end of Table I block 2 or vowels with macron plus breathing and accent like 'ὂ̄' found in only one book [7] reproducing a Greek text of the early 5th century B.C. and simulating the long vowels 'η' and 'ω' missing in that dialect. These characters are produced with macros such as \mc or \br for 'macron' or 'brève' with three parameters, the third being the diacritized vowel:

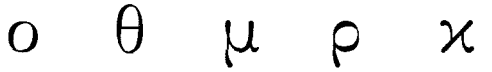
```
\mc>~o     gives  ὦ̄        \mc<’e   gives  ἕ̄
\br\ >o           ὂ̆        \br<~e          ἑ͂̆
```

When such characters appear frequently (in reference [7] there are several in each line) they could be placed in a new adjunct font accessed by the 'macron' and 'brève' symbols. For only 'ε' and 'o' with macron there are 24 characters which could be fitted in the present adjunct font.

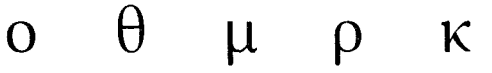
The fonts. In the multitude of Greek fonts with variable stroke width and most satisfying to Greek readers, we may distinguish two prevailing styles: one is the style of fonts called 'simple' (*ἁπλὰ*) in Greece, which we shall venture to call *Didot style* as it has a distinct relation to Didot's fonts and has probably evolved from them; the other of fonts tending generally to what is called *Elsevier* or *Times.* The *Didot* style is still amazingly close to the fonts used by the famed Parisian firm of Didot (18th – 20th century). François Ambroise Didot (1730–1804), apparently influenced by John Baskerville (1706–1775), created the *Didot* fonts in 'new style roman' (which in turn influenced Giambattista Bodoni, 1740–1813). He was said by Benjamin Franklin Bache (apprenticed to Didot by his grandfather Benjamin Franklin) to be 'the best printer of this age and even the best that has ever been seen'. Firmin Didot (1764–1836) created improved fonts, and Ambroise Firmin Didot (1790–1876) printed many Greek books, among them several of the eminent Greek scholar A.Koraís in Paris, and sent printing presses and hundreds

of books to Greek towns even before the start of the Greek War of Independence (1821) [8]. Other printing presses were also brought to Greece from England and Italy, notably by Stanhope to Missolonghi, but *Didot* style fonts seem to have been generally adopted in Greece. A slight change in the fonts of Koraís' books between 1807 and 1823 might possibly indicate the shift from François Ambroise to Firmin Didot fonts.

Didot type fonts are used in older Greek publications, in the government gazette (reminiscent of the font used in the first official printing of the Provisional Constitution of Greece enacted in Corinth in 1822), in several historical and philological treatises, in the Teubner classical editions (though more ornate), and in TEX as Greek math symbols. The *Elsevier* or *Times* types of fonts are closer to the 'roman' style, and are mostly used in scientific books, novels and newspapers.



Didot style (almost Levy's [1])



Near *Times-Elsevier* style (*Euclid* font)

Some of the many differences between the two styles stand out as distinguishing characteristics: in *Didot* characters the thicker parts of curved lines are those sloping up toward the right (which happens in the letter 'O' when its internal oval is tilted toward the right) as in the first row in the figure above, whereas in the *Times-Elsevier* the thicker arcs are slightly sloping upward toward the left. The *Didot* have arms which get thicker toward their ends, a lowercase 'kappa' resembling an 'x', and the legs of 'mu' or 'rho' or of both generally curving toward the right, whereas the *Times-Elsevier* style has straight arms mostly of constant thickness and a lowercase 'kappa' resembling the uppercase. Both types can be upright or slanted, though some editions of the classics have slanted lowercase with conspicuous upright uppercase characters. Levy's fonts, as well as Haralambous' and Thull's are clearly related to the *Didot* upright. We wanted a style approaching the Times-Elsevier, which did not exist in TEX, but with some aesthetic changes as well as practical ones, such as a preference for horizontal and vertical lines which print better at medium resolutions.

The character files in *Computer Modern Typefaces* and the corresponding character figures were of immense help, as were those of Levy, and served as models for ours. Without them our work would have been much harder. However, all character programs had to be redone to the new style, even those of the 14 uppercase letters similar in Greek and English and of some digits and punctuation marks, though their changes were minor. The lowercase characters presented the greatest difficulty, as they have quite different curves from the English and several intersecting or convoluted branches. The aim was to make a font pleasant and acceptable to readers accustomed to present day Greek publications, and causing no hesitation or delay for querying or appreciating unusual strokes.

One old ligature was re-introduced as a character in the Greek alphabet, after a search and comparison, necessarily inexpert, in Byzantine manuscripts of Mount Athos and of Patmos, and in early editions of Greek texts, such as the Estienne edition of the Homeric poems (Paris 1566), the Cambridge edition of the Suidas Lexicon (1705), and in more recent publications. It is a pity that some of these beautiful characters cannot be fitted into today's rather rigid font styles. The new character is a ligature of omicron 'ο' with upsilon 'υ' over it, making an infinity sign with the open end uppermost: 'ϑ, ϴ'. It first appears in a manuscript of about 512 AD and regularly after about 880 AD [9], and was used in renaissance editions of the classics and later up to about 200 years ago. Some people, including one of the authors, use it in handwriting today. It may be remembered from Byzantine icons of Mary "MOTHER OF GOD", i.e., "MHTHP ΘEOY", abbreviated by iconographers to "MP Θϴ".

We assigned this character to the key 'v, V' instead of accessing it as a ligature of 'o' and '" because some people will not want the diphthong 'μζ οψ' to turn up automatically as a '". Separate characters had to be made of '" with every accent and placed in the main font, and with accent-breathing in the adjunct font, so there are now eight vowels 'α ε η ι ο υ ο ω ϑ' with all corresponding diacritics. This character is presented to former colleagues in Engineering and Applied Math at Brown University who had been inquiring about additional Greek characters for use as symbols of newly introduced quantities. It also affords a significant shortening of Greek text, as the 'οψ' diphthong may occur quite frequently, as in the following line

Αυτούς τους κουφούς ανθρώπους
Αυτός τϑς κϑφός ανθρώπϑς

TABLE III
THE NEW GREEK FONTS

Font gm10
ΑΒΓΔΕΖΗΘΙΚΛΜΝΞΟΠΡΣΤΥΦΧΨΩϑ αβγδεζηθικλμνξοπρσςτυφχψωϑ

Font gmb10
ΑΒΓΔΕΖΗΘΙΚΛΜΝΞΟΠΡΣΤΥΦΧΨΩϑ αβγδεζηθικλμνξοπρσςτυφχψωϑ

Font gmss10
ΑΒΓΔΕΖΗΘΙΚΛΜΝΞΟΠΡΣΤΥΦΧΨΩϑ αβγδεζηθικλμνξοπρσςτυφχψωϑ

Font gmssb10
ΑΒΓΔΕΖΗΘΙΚΛΜΝΞΟΠΡΣΤΥΦΧΨΩϑ αβγδεζηθικλμνξοπρσςτυφχψωϑ

Font gmsl10
ΑΒΓΔΕΖΗΘΙΚΛΜΝΞΟΠΡΣΤΥΦΧΨΩϑ αβγδεζηθικλμνξοπρσςτυφχψωϑ

Font gmslb10
ΑΒΓΔΕΖΗΘΙΚΛΜΝΞΟΠΡΣΤΥΦΧΨΩϑ αβγδεζηθικλμνξοπρσςτυφχψωϑ

Font gmssl10
ΑΒΓΔΕΖΗΘΙΚΛΜΝΞΟΠΡΣΤΥΦΧΨΩϑ αβγδεζηθικλμνξοπρσςτυφχψωϑ

Font gmsslb10
ΑΒΓΔΕΖΗΘΙΚΛΜΝΞΟΠΡΣΤΥΦΧΨΩϑ αβγδεζηθικλμνξοπρσςτυφχψωϑ

Font gc10
ΑΒΓΔΕΖΗΘΙΚΛΜΝΞΟΠΡΣΤΥΦΧΨΩϑ ΑΒΓΔΕΖΗΘΙΚΛΜΝΞΟΠΡΣΤΥΦΧΨΩϑ

Font gcss10
ΑΒΓΔΕΖΗΘΙΚΛΜΝΞΟΠΡΣΤΥΦΧΨΩϑ ΑΒΓΔΕΖΗΘΙΚΛΜΝΞΟΠΡΣΤΥΦΧΨΩϑ

Font gcsl10
ΑΒΓΔΕΖΗΘΙΚΛΜΝΞΟΠΡΣΤΥΦΧΨΩϑ *ΑΒΓΔΕΖΗΘΙΚΛΜΝΞΟΠΡΣΤΥΦΧΨΩϑ*

Font gcssl10
ΑΒΓΔΕΖΗΘΙΚΛΜΝΞΟΠΡΣΤΥΦΧΨΩϑ *ΑΒΓΔΕΖΗΘΙΚΛΜΝΞΟΠΡΣΤΥΦΧΨΩϑ*

Three additional symbols were designed to complete the Greek numerals, which run as follows

	α′ 1		ι′ 10		ρ′ 100
	β′ 2		κ′ 20		σ′ 200
	γ′ 3		λ′ 30		τ′ 300
	δ′ 4		μ′ 40		υ′ 400
	ε′ 5		ν′ 50		φ′ 500
stigma	ϛ′ 6		ξ′ 60		χ′ 600
	ζ′ 7		ο′ 70		ψ′ 700
	η′ 8		π′ 80		ω′ 800
	θ′ 9	koppa	ϟ′ 90	sampi	ϡ′ 900

The regular alphabetic Greek numerals are entered as letters of a regular Greek font with the last one primed. The three additional symbols for 6, 90, 900 are in the adjunct fonts and are entered as `\s \n \p`, giving: '`ϛ ϟ ϡ`'. Thousands are represented with the same symbols and a low left subscript entered as `\x` before the first digit of the numeral. E.g., `\x a'` gives ͵α′ (1000), `\x\p '` gives ͵ϡ′ (900,000), and `\x a\p\n a'` gives ͵αϡϟα′ (1991). If the *Didot* style is prefered for numerals, the Levy font could be called, as was done for the two α's of the greek numeral for 1991 which look more archaic. Finally a digamma '`F`' (entered as `\f`) was placed in the main fonts to satisfy classicists, and the new symbol '`ϐ`' (entered as `C`) was introduced for representing the Greek drachma (δραχμή) in analogy with the $-sign. Uppercase Greek numerals for epigraphic use have been given by I. Haralambous [4].

The same character programs were used with suitable parameter files to create a regular font, a bold, a sans serif and a sans serif bold font, as well as the corresponding four slanted fonts, all in magsteps 0, .5, 1, 2 and 3. They are 128-character Modern Greek fonts with names starting with GM, and each has a "classical" adjunct font with a corresponding name starting with GP. In addition, four *Caps-Small-Caps* fonts were made with names starting with GC at the same magnifications, as well as four fonts of 9, 8, and 7 points for Greek subscripts in math mode. All are listed below with, hopefully, recognizable provisional names, which can later be changed to conform with a standardized font-naming scheme as described by Karl Berry [10].

gm10 gmb10 gmss10 gmssb10

gp10	gpb10	gpss10	gpssb10
gmsl10	gmslb10	gmssl10	gmsslb10
gpsl10	gpslb10	gpssl10	gmpslb10
gc10	gcsl10	gcss10	gcssl10
gm9	gmb9	gm8	gm7

The fonts of the top row at zero magnification are entered with `\mz`, `\mbz`, `\mssz`, `\mssbz`. For magnification 1 the last 'z' changes to 'un', and for magnification 2 to 'tw', provided all these fonts have been defined. TEX 3 accepts these extra fonts without difficulty, but earlier versions may give the error message "`! TeX capacity exceeded, sorry.`" calling for an increase of the font memory. The various fonts are shown in Table III, and samples of Greek text in the page following it.

We give the name of *Euclid* to these fonts not so much for the shortest paths of the arms of the characters, which they mostly follow, being straight, but in honor of an earlier Euclid, archon of Athens in 403 B.C. when the Ionic form of the Greek alphabet was officially accepted, i.e. the greek alphabet as we know it today.

Hyphenation. English hyphenation leads to many in Greek text. Greek hyphenation rules are simple and have been discussed [4] and hyphenation patterns have been announced [12], but they may not fully apply to Classical Greek. We are working on hyphenation patterns for strictly Greek text, using the initial version of PCTEX 3.0 [11] without multilingual capabilities. With enhanced hyphenation capabilities these patterns should work also for mixed language texts.

The Greek Monitor Characters. The word processor PC-Write [13], mentioned years ago in the *TUGboat* as using only ASCII characters hence suitable for writing and editing TEX files, and used for the last four years by one of the authors, came out recently in a shorter version, PC-Write Lite, capable of using foreign fonts both onscreen and for printing, and with the facility for creating the monitor fonts on a 8×14 pixel matrix. We used it to make the set of simple Greek letters in the displayed screen photograph. These characters are defined in a new edit control file `ED.GRK`, where they are given positions above the regular 128 ASCII characters and are assigned to the corresponding English keys as in Table II. As used presently, every text file `<file>.grk` meant to display Greek characters onscreen must have the extension `.grk` which directs PC-Lite to select the edit control file `ED.GRK`. At present the `<Caps-Lock>` key shifts

Greek screen characters designed with PC-Write Lite.

between English and Greek characters during word processing and in case of errors `<ALT-^>` transcribes the letter before the cursor. To print Greek from the word processor one should match or transcribe the positions of the screen characters to those of the printer fonts he acquires.

The major problem remained, however, of how to TEX a file with Greek characters in positions above the 128th ASCII when all Greek TEX characters are below the 128th. The problem was solved with the kind help of Mr.Mark Zehr of Quicksoft, who patiently instructed us on how to create a new printer control file for transcribing the Greek file into an English one with the character correspondence of Table II. We call this file `PR.TEX` and rename it `PR.DEF` for making the transcription. Furthermore a so-called 'dot-command' must be written at the top of `<file>.grk` for directing the output to `<file>.tex`

```
<ALT-G>.0:<file>.tex
```

Pressing `<ALT-@>` from the open file in PC-Lite, transcribes the Greek `<file>.GRK` to the English `<file>.TEX` of the same name but a `.TEX` extension. When TEXed this gives the proper Greek or English text as specified by the font commands. This method works with all TEX versions.

However, we also found another solution usable with TEX 3. We made a Greek-to-Latin character substitution file `grk_eng.tex` which is called by `greekmac` and operates automatically during TEXing of the Greek `<file>.grk`, independently of the word processor, remaining harmlessly unused with the English characters. It requires TEX commands to be in English because TEX reads them before the character substitution and does not recognize them in Greek. In TEXing one should now include the extension `.grk`

```
tex <file>.grk   or
tex &greek <file>.grk
```

Seeing Greek characters on the screen makes reading easier and more pleasant, and helps prevent all the mistypings of the characters with a 'not obvious' English correspondence (Table II).

If hard copy of the (un-TEXed) Greek file is desired and Greek printer fonts are not available, one should be content with the English transcription which can be printed without actually creating the file `<file>.tex`. Without the dot line ".0:..." in the text file, the pressing of `<ALT-@>` directs the transcription of `<file>.GRK` to the printer (provided `PR.TEX` has been renamed `PR.DEF`).

Summary and Conclusion. We presented a new set of Greek fonts with all the required characters for Modern and Classical Greek, in regular, bold, sans serif, sans serif bold, upright and slanted, and the corresponding caps-small-caps fonts, each placed in a pair of 128-position groups; and a system which accesses both groups as if they were a single 256-character font, though without kerning between characters in different groups. The system can be easily extended to use three 128-position groups or, with 256-code systems, two or more 256-position groups of characters as if they were one large font, provided each group is composed of characters with a common trait which may be used for shifting access to that group. This method of shifting to adjunct fonts should be useful for languages with large numbers of characters.

With 256-position fonts it would seem advantageous to place all characters and symbols of uniaccent Greek in the lower 128 positions, where they may then be easily separated into a 128-position Modern Greek font for the less fortunate users of the smaller size fonts. Several empty lower-128 positions may be filled with classical Greek characters such as accented vowels (without breathing). Vowels with breathings and other symbols of Classical Greek or lexicography would seem better placed in the upper 128 positions, to be easily converted into a 128-code adjunct font for writing Classical Greek even with the smaller size fonts.

Conversely with the system of virtual fonts one could join our main and adjunct font into one 256-position font, or select 256 characters for any particular application from several 128- or 256-position adjunct fonts, with kerning even between characters of different original fonts.

The purpose of this artacle is to present the work we have done and the system of adjunct fonts we have devised. Our coding scheme is described in order to show how this system makes possible the use of two 128-position fonts as if they were one 256-position font, and not as a candidate for standardization, though we hope that it shows some useful possibilities for future standardization work. It should be kept in mind, however, that a suitable transliteration program similar to the one we presented could automatically match a particular coding to an established standard.

Remarks. METAFONT is an excellent font-making system permitting the precise creation of characters of any design, and we were delighted to use it. The individual fontmaking steps are reasonably rapid but must be repeated so many times that the whole font-making process becomes quite long. Even the use of a RAM-disk containing all necessary files does not speed-up the process significantly. The acceptance of each character and of the font as a whole is purely a matter of aesthetics, and requires repeated changes in the program file followed by judging the display of the enlarged character. This cycle is repeated some 10 to 20 times for each character, even more with complex characters like 'ξ', a process lasting from about a half to two days for each character, less for the same character in a different font. The satisfactory character is then formed in proofmode and printed, only to be found awry or deficient in some details. A new cycle of corrections follows and usually produces a print which survives only a few hours before crying faults are discovered. The cycle is repeated with decreasing frequency until the proof survives a week or two. New changes are then made for certain groups of characters or when a macro of wider application is introduced, or when forming accents and breathings and making the macros for their precise positioning over each vowel. After all characters are done, the font is created with METAFONT, preferably at a few magnifications, each taking from 9 to 12 minutes on a PC AT compatible at 8MHZ, and the uniformity and compatibility of the characters is checked and adjusted after several printouts, each ending with a re-creation of the corrected font. In particular the characters φ ψ ω required special attention and re-designing in order to come out symmetric on an HP Laserjet IIP printer (early attempts on a 180DPI dot matrix printer were hopeless). Finally the kerning is adjusted in several steps from spacing estimates made on printouts of matrices containing all possible character pairs, each step requiring a new fontmaking. The whole process, together with

learning the tricks of METAFONT, lasted almost two year (which is not bad – not referring to quality – in comparison with the years taken by Louis XIV's commission to design the Royal Alphabet [14] or with the duration of the *"Euler Project"* [15]), but additional adjustments have been postponed until additional kerning is available. Here are some of the thoughts and wishes which kept recurring during this long process.

1. METAFONT does not use the coprocessor for calculating points and curves and pixels, but works with some internal process of its own. Would the use of the coprocessor speed things up, and how could this be achieved?
2. Could an extension of METAFONT be made to recalculate only the altered parts of a character and, even better, show the change on the displayed character?
3. Could an extension of METAFONT remake single characters or adjust kerns without recreating the full font?

Acknowledgment. We wish to thank several friends who helped us. Without Barbara Beeton's great help with advice and information this project would not have started. Professor S. Levy's kind offer of all program files of his excellent Greek system and fonts made it possible for us to start our system at an advanced stage. Professor Alan Boegehold of the Classics Department at Brown University and presently chairman of the Managing Committee of the American School of Classical Studies at Athens read the original draft and advised us on sources of ancient scripts. His criticism led to changes which significantly simplify the typesetting of Greek. David Durand kindly helped one of the authors correct several errors, and Elli Mylonas, project manager of the *PANDORA* program [2], did the search for diacritized vowels in the classical literature. We are grateful to all.

References

1. Levy, S., Using Greek Fonts with TEX, *TUGboat,* **9**(1), 1988, 20-24
2. As indicated by a preliminary computer search of the index of words in the Greek literature from Homer to the 6th century AD on the CD ROM of the *Thesaurus Linguae Graecae* (UC-Irvine). The search for *diacritized* vowels was done with the program *PANDORA 2.3* produced under the auspices of the *PERSEUS* project at Harvard U.
3. An acute single accent is used in the book Ἑλληνική Μυθολογία (Greek Mythology), Athens 1986, of Professor I.Kakridis, early proponent of the single accent and hero of the (in-)famous 'Trial of the Accents' in the forties in Athens.
4. Haralambous, Y., and Thull, K., Typesetting Modern Greek with 128-character codes, *TUGboat,* **10**(3), 1989, 354-9
5. Haralambous, Y., On TEX and Greek..., *TUGboat,* **12**(2), 1991.
6. Fulling, S.A., Where's the Greek Shift Key?, *TUGboat,* **11**(3), 1990, 371-2
7. Found only in Willetts, R.F., *The Great Lawcode of Gortyna*, Oxford Un.Press, 1967.
8. Koraís' books and products of Didot's printing presses in Greek towns exist at the Gennadeion Library, American School of Classical Studies at Athens. See also: Walton, R.W., *The Greek Book 1476-1825,* Tenth Int. Congress of Bibliophiles, Athens Sept. 30, 1987.
9. Gardthausen, V., *Griechische Palaeographie*, Veit, Leipzig, 1911. Vol.2, Tables 1-12.
10. Berry, K., Filenames for fonts, *TUGboat,* **11**(4), 1990, 517-20.
11. Personal TEX, Inc., 12 Madrona Ave., Mill Valley, CA 94941
12. Haralambous, Y., Typesetting Modern Greek – An Update, *TUGboat,* **11**(1), 1990, 26.
13. Quicksoft, Inc., 219 1st Ave N # 224, Seattle, WA 98109-9911
14. Knuth, D.E., *Mathematical Typography,* Bull. of the Am. Math. Soc., v.1, n.2, March, 1979., and *TEX and* METAFONT, Am. Math. Soc., 1979.
15. Siegel, D.R., *The Euler Project at Stanford*, Dpt. of Computer Science, Stanford Univ., 1985.

First submitted: 2-16-91. In final form: 12-21-91.

◇ C.Mylonas
Κουντουριώτη 5
14563 Κηφισιά
Athens, Greece

◇ Ron Whitney
TEX Users Group
rfw@Math.AMS.com

or at times:
Engineering, Brown University
Providence, R.I. 02906

SAMPLE TEXTS in GM10 and GMSSL10.

Τὴ γλώσσα, αὐτὸ το σπαρταριστὸ ἀπὸ ζωὴ δημιούργημα τοῦ ἀνθρώπινου πνεύματος, δὲν τὴν κατασκευάζουν οἱ γλωσσολόγοι, οὔτε οἱ δάσκαλοι στὰ σχολειά. Τὴν γλώσσα τὴν πλάθει πρῶτα ὁ λαὸς καὶ μὲ τὴν καθημερινή του ὁμιλία, ἀλλὰ καὶ μὲ τὶς εὐγενέστερες ἐκδηλώσεις του, ὅπως π.χ. τὸ δημοτικὸ τραγούδι. Τὴν πλάθουν ὅμως καὶ κορυφαῖοι εἴτε στὴν ἔκφραση αὐστηρὰ λογικῶν διανοημάτων (μαθηματικῶν, νομικῶν) εἴτε στὴν ἔκφραση συναισθημάτων. Τὴν πλάθουν καὶ οἱ δυὸ αὐτοὶ ὅταν ἔχουν καὶ αἰσθητικὸ αἰσθητήριο εἴτε βοηθημένοι ἀπὸ τὴν καθαρότητα τῆς λογικῆς των σκέψης, εἴτε ἀπό το χάρισμα ποὺ τοὺς δόθηκε μὲ λέξεις, ποὺ πάντα ἔχουν καὶ ἕνα λογικὸ νόημα, νὰ ἐκφράζουν τὸ ἄλογο. Γι' αὐτὸ ὅταν θέλωμε νὰ καταλάβωμε το πνεῦμα μιᾶς γλώσσας δὲν θὰ καταφύγωμε στὶς γραμματικὲς καὶ τὰ συντακτικά, ἀλλὰ στὰ κείμενα τῶν μεγάλων συγγραφέων. Αὐτοί ὑπαγορεύουν στοὺς γλωσσολόγους τοὺς νόμους τῆς γλώσσας καὶ ὄχι οἱ γλωσσολόγοι στοὺς δημιουργοὺς τῆς γλώσσας.

From K.Tsatsos: Glossa kai Ethnos, Efthini, 112, 1981, 145-8.

***Γωνιακή ταχύτητα συναρτήσει των γωνιών τσ* Euler.** *Το στερεό περιστρέφεται με τοπικό σπιν* $\dot{\psi} = \dot{\psi}\mathbf{k}''$ *στο πλαίσιο* $Ox''y''z''$ *(Σχ. 15.2) πσ περιστρέφεται με ένα μετοχικό ρυθμό κλονισμό* $\dot{\theta} = \dot{\theta}\mathbf{k}'$ *ως προς το πλαίσιο* $Ox'y'z'$*, πσ κι' αυτό περιστρέφεται με ένα μετοχικό ρυθμό μεταπτώσεως* $\dot{\varphi} = \dot{\varphi}\mathbf{K}$ *ως προς το* $Oxyz$*. Τα* $\dot{\varphi}, \dot{\theta}, \dot{\psi}$ *είναι διανυσματικές γωνιακές ταχύτητες πσ με διαδοχικές συνθέσεις ως τοπικής και μετοχικής δίνσν την ολική γωνιακή ταχύτητα* $\boldsymbol{\omega}$ *τσ στερεό*

$$\boldsymbol{\omega} = \dot{\boldsymbol{\varphi}} + \dot{\boldsymbol{\theta}} + \dot{\boldsymbol{\psi}}.$$

C.M.: MECHANICS II,Rigid Body Dynamics, 2nd Ed.Athens, 1984.

	Density	Total
Strain Energy:	$w(\varepsilon_{ij}^t) = \int_0^{\varepsilon_{ij}^t} \sigma_{ij} d\varepsilon_{ij},$	$W = \int_V w(\varepsilon_{ij}^t) dV$
Complementary Energy:	$\text{ϑ}(\sigma_{ij}^t) = \int_0^{\sigma_{ij}^t} \varepsilon_{ij} d\sigma_{ij},$	$\text{Ϙ} = \int_V \text{ϑ}(\sigma_{ij}^t) dV$

TABLE IV

Font GM10 x 1.44 Adjunct Font GP10 x 1.44

'0	'1	'2	'3	'4	'5	'6	'7		'0	'1	'2	'3	'4	'5	'6	'7
ὅ	ά	έ	ή	ί	ό	ύ	ώ	'00x	ᾀ	ᾁ	ᾐ	ᾑ	ᾠ	ᾡ	ᾲ	ᾳ
ὃ	ὰ	ὲ	ὴ	ὶ	ὸ	ὺ	ὼ	'01x	ᾄ	ᾅ	ᾔ	ᾕ	ᾤ	ᾥ	ᾴ	ᾷ
ᾶ	ᾱ	ᾶ	ῆ	ῖ	˘	¯	ῶ	'02x	ᾆ	ᾇ	ᾖ	ᾗ	ᾦ	ᾧ	ῂ	ῃ
—	·̔	·̓	ΰ	Ϊ	’	ῦ	ὃ	'03x	ᾈ	ᾉ	ᾘ	ᾙ	ᾨ	ᾩ	ῄ	ῇ
F	!	–	ΐ	ϊ	%	″	′	'04x	ᾊ	ᾋ	ᾚ	ᾛ	ᾪ	ᾫ	ῒ	ΐ
(	)	*	+	,	-	.	/	'05x	ᾌ	ᾍ	ᾜ	ᾝ	ᾬ	ᾭ	ῗ	ῧ
0	1	2	3	4	5	6	7	'06x	ᾎ	ᾏ	ᾞ	ᾟ	ᾮ	ᾯ	ῢ	ῠ
8	9	:	·	ΐ	=	Ẏ	;	'07x	ᾐ	ᾑ	ᾠ	ᾡ	ᾲ	ᾳ	ῲ	ῳ
@	A	B	δ	Δ	E	Φ	Γ	'10x	ῥ	A	A		ῥ	E	H	Ω
H	I	Θ	K	Λ	M	N	O	'11x	H	I	ς	Ϛ	ϗ	Ϙ	ϐ	O
Π	X	P	Σ	T	Y	ϑ	Ω	'12x			P			Y	ϑ	Ω
Ξ	Ψ	Z	[	§	]	ü	˙	'13x	ᾶ	ᾱ	ῆ	ῑ	ὃ	ῦ	ῶ	ὅ
`	α	β	σ	δ	ε	φ	γ	'14x	`	α				ε		
η	ι	θ	κ	λ	μ	ν	o	'15x	η	ι						o
π	χ	ρ	ς	τ	υ	ϑ	ω	'16x			ρ			υ	ϑ	ω
ξ	ψ	ζ	«	‚	»	˜	¨	'17x					‚	‚	˜	

Comments on "Filenames for Fonts" (*TUGboat* 11, no. 4)*

Frank Mittelbach

Abstract

In *TUGboat* 11, no. 4, a nomenclature for font files was proposed by Karl Berry. I disagree with Berry's proposal in some important points and would like to put these in writing in the hope that we will find some suitable agreement before anything is adopted as a standard.[1] I was aware of an ongoing email discussion about this topic but unfortunately didn't pay enough attention to realise that this would lead at this stage to a definite proposal.

Although this article points out several possibilities, it is not meant as a counter proposal. It is written in the hope that re-opening the discussion will lead to the best possible solution. In its current state, Berry's proposal cannot be used for LaTeX 3.0 (cf. sections 2 and 4) and this means that the majority of TeX users will be forced to use something different.[2] Thus, however consistent and rational it may be, his scheme can never become a universal standard.

1 Identifying font characteristics

Berry said that his proposal follows and simplifies the scheme we adopted for the new font selection scheme of LaTeX [11]. But in my opinion it makes the same mistake as we did in our first proposal for a new font selection scheme for TeX fonts [10]. The main idea behind identifying certain properties of fonts individually is the desire to change them independently. If, for example, a designer defines the layout of a heading to appear in 'bold extended' typeface, then a part of this heading that is to be emphasized should appear in a corresponding font, preferably in 'bold extended italic' or at least in 'bold italic'. This is possible if one identifies 'bold' as the *weight*, 'extended' as the *width* and 'italic' as the *shape* or *variant* of the current font.

However, Berry identifies both 'sans serif' and 'typewriter' as *variants*, whereas we think that these are invariants of a font family and consequently should appear in the font family name. The reason for this decision is the fact that there is practically no font family which consists of both a 'serif' and a 'sans serif' variant or which contains an additional 'typewriter' variant. We do not view the Computer Modern Family as a counter example: it is a meta-family consisting of several independent font families which are only loosely connected by design principles. Otherwise one has to accept Concrete Roman as part of this family[3] and this seems a bit far-fetched.

2 The Computer Modern families

It is true that more and more fonts are becoming accessible through TeX and that it is therefore time to introduce a naming convention which allows them to be handled in a consistent manner. However, the main fonts in use are still public domain fonts generated by METAFONT. This is because, firstly, they cost nothing (or almost nothing) and, secondly, they ensure compatibility since most of them are included in the standard TeX distributions. For these reasons a scheme that does not cover these fonts is only of limited use. Berry never mentions how Computer Modern by Knuth [7] or Pandora by Billawala [2] could be integrated into his scheme.[4]

3 Font names of eight characters

As Berry correctly states, eight characters are definitely not enough to cover all possible font families with all their variations, at least not if verbose naming is used. However, if we encode the font names into arbitrary sequences of letters and

* The assistance of Chris Rowley is acknowledged with pleasure.

[1] The re-implementation of LaTeX will allow the user to access a broader range of fonts and it would be a big disadvantage if the method used implements a standard different from the one used in other macro packages.

[2] At the moment, only comparatively few users are in a position to actually use the new typefaces and most therefore have to rely on Computer Modern or a virtual variant thereof (implementing a standard TeX encoding).

[3] The Concrete Roman family is constructed by using the Computer Modern METAFONT sources and applying new parameter sets. In our notation the family 'concrete roman' consists of the *variants* 'normal', 'italic' and 'small caps', all in medium *width* and *weight*. Additionally a 'slanted' or 'sloped' *variant* in 'condensed' *width* exists. Examples of this typeface can be found in [11].

[4] While it might be possible to come up with some two-letter combinations for the typeface names and perhaps 't' (i.e. TeX distribution) for the foundry, there is no possibility in Berry's scheme of including virtual fonts that extend METAFONT fonts to 256-character codepages, cf. section 4.

numbers (beginning with a letter) then we can address $26 \times 36^7 = ?$ different fonts.[5] I suppose that nobody would like to remember Times-Roman as `z5zcvp49`, so perhaps a more readable encoding has to be found, but we should not choose a system that already has difficulties in covering the current range of available fonts.

However, depending on the addressing method used within TEX, the actual names of the files can be of minor importance since they need concern only format developers. For example, in the new font selection scheme for LATEX [11], the user will specify fonts by characteristics which consist, in principle, of arbitrarily long strings.[6] At the most primitive level, this user-interface consists of command sequences such as

```
\family{tim}\series{bc}%
\shape{sc}\selectfont
```

this loads a small caps variant of Times-Roman in weight/width bold condensed (in the current size). This might indeed correspond to some external file named `z5zcvp49`, without forcing the user to learn this fact. To use such a scheme successfully we have to ensure that there no longer exist situations where the user is forced to return to LATEX's `\newfont` command or TEX's `\font` primitive. In particular, this requires proper documentation[7] of the available fonts in the form of tables for `\family`, `\series`, etc., together with the ability to access fonts in arbitrary sizes since many fonts can be scaled nowadays by the output devices. This important feature will be added to the new font selection scheme in the near future; the implementation is currently under way.

The following sections deal with individual parts of Berry's proposal. They are nothing more than observations and do not add up to a new proposal for using the available number of characters.

[5] Similar considerations apply to the case of user names on length-restricted systems. While `pzf5hz` is perhaps difficult to connect with "Mittelbach", this approach allows over 60 000 EDS employees access to the company's net without conflicts.

[6] It should be pointed out that the new font selection scheme is independent of LATEX and could therefore serve as a new standard, just like the old one proposed by Knuth [9, pp. 414–415] at a time when only a few fonts were available and subsequently implemented in most macro packages.

[7] A task that has still to be undertaken for LATEX 3.0: any volunteers?

Denoting the foundry While it seems nice to have names that are easy to remember[8], I have some reservations about the foundry table given in [1, p. 517]. It might be possible to cover the major foundries of the western world, but the market is young and will certainly expand enormously in the near future.[9] The present list is nowhere near complete: where, for example, are Monotype and Linotype?

Denoting the typeface family The table of typeface families reads like a PostScript brochure. While at present this is certainly an important source of non-METAFONT fonts used with TEX, one should look closely at all the other families provided by the major printer companies to see whether or not they fit into any proposed scheme.[10]

Denoting the weight and expansion Given the constraints on available characters for use in the font names, I would suggest squeezing this information into one character. One can probably use 'memorable' characters for the most usual combinations and assign the remaining characters to all other combinations. To reduce the number of possible combinations one should drop the distinction between human and automatic scaling in expansion. While this is an interesting fact, I doubt whether any foundry supplies the same typeface in both ways.

Denoting the variant My only concerns here are those I expressed earlier (see section 1) about what constitutes a variant.

Denoting the size To avoid the problem of unspecifiable font sizes, I suggest the use of a two-digit hexadecimal (or even base-36) number. For standard sizes, i.e. those in the range 5pt to 20pt, this is as readable as a decimal number; and for the usual display sizes (e.g. 72pt) one would surely get used to it. This would also allow the packing of additional information into this part of the font name, as explained in the next section.

[8] Whatever this means when only a single character is to be used.

[9] Since it is already difficult to assign appropriate letters, one might think of dropping this approach completely to avoid giving certain foundries preference over others.

[10] There exist by now tools to generate `.tfm` files from down-loadable fonts for many font formats in different printer languages.

4 Font Encoding Schemes

Berry mentions the problem of virtual fonts. It is in principle possible to generate arbitrary fonts by combining characters from different typefaces into one virtual font. While this method allows the creation of an unlimited number of fonts and could certainly blow up any scheme, it seems questionable whether this will actually happen. A natural usage of this concept would be to add certain missing characters or symbols to a font so that it can be used with a standard macro package. In such a case, however, the resulting virtual font would still be clearly identifiable by its major raw font. On the other hand, virtual fonts could completely dispose of the problem of font encoding, provided that the TEX community can agree on a few standard layouts for 'latin' (cf. [3]), 'math', etc.

The use of '`r`' for raw `.tfm` files, as pointed out by Berry, works only for fonts which have no design size and this again rules out any font produced with METAFONT, since virtual fonts for such families (following the coding scheme as proposed in [3])[11] cannot be specified within Berry's naming conventions. I therefore suggest coding this information within the design size, by adding a suitable number to the actual design size to indicate that a raw `.tfm` file is to be used.[12] If the design size were coded in hexadecimal notation, this would allow design sizes up to 127pt (`"00`–`"7F`) for the (virtual) fonts which are actually used (and which have a standard TEX encoding scheme), the accompanying range (`"80`–`"FF`) being left for raw `.tfm` files.

[11] To my knowledge, this work (for the Computer Modern families) has already been undertaken in Germany and is at the moment in β-testing.

[12] The use of unusual font sizes for the raw `.tfm` seems appropriate since these font metric files are of interest only to those who have to set up virtual fonts.

References

[1] Karl Berry. Filenames for fonts. *TUGboat*, 11(4):517–520, 1990.

[2] Nazneen N. Billawala. Metamarks: Preliminary studies for a Pandora's Box of shapes. Technical Report STAN-CS-89-1256, Stanford University, Department of Computer Science, Stanford, California 94305, 1989.

[3] Michael J. Ferguson. Report on multilingual activities. *TUGboat*, 11(4):514–516, 1990.

[4] Donald E. Knuth. Computers & Typesetting. Addison-Wesley, Reading, Massachusetts, 1986. Consists of [9, 8, 6, 5, 7].

[5] Donald E. Knuth. *METAFONT: The Program.* Volume D of *Computers & Typesetting* [4], 1986.

[6] Donald E. Knuth. *The METAFONTbook.* Volume C of *Computers & Typesetting* [4], 1986.

[7] Donald E. Knuth. *Computer Modern Typefaces.* Volume E of *Computers & Typesetting* [4], July 1987. Reprint with corrections.

[8] Donald E. Knuth. *TEX: The Program.* Volume B of *Computers & Typesetting* [4], May 1988. Reprint with corrections.

[9] Donald E. Knuth. *The TEXbook.* Volume A of *Computers & Typesetting* [4], May 1989. Eighth printing.

[10] Frank Mittelbach and Rainer Schöpf. A new font selection scheme for TEX macro packages—the basic macros. *TUGboat*, 10(2):222–238, July 1989.

[11] Frank Mittelbach and Rainer Schöpf. Reprint (with corrections): The new font family selection — user interface to standard LATEX. *TUGboat*, 11(2):297–305, June 1990.

◇ Frank Mittelbach
Electronic Data Systems
(Deutschland) GmbH
Eisenstraße 56 (N 15), D-6090
Rüsselsheim, Federal Republic
of Germany
Tel. +49 6142 803267
mittelbach@mzdmza.zdv.uni-mainz.de

Output Devices

The DVI Driver Standard, Level 0

The TUG DVI Driver Standards Committee

Abstract

The TUG DVI Driver Standard defines functional and interface requirements for computer programs (DVI processors) that read and translate files in the DVI page description language. This document is the subset of the DVI standard (level 0) applying to minimally functional DVI processors. The specifications here should be considered minimal; developers are encouraged to write drivers exceeding these specifications.

(The version of the Level 0 Standard presented here is draft 0.05. It has been reviewed by the TUG DVI Driver Standards Committee and is now being presented to the TUG membership at large for review.)

The complete standard will be presented as a series of tiers requiring increasingly stringent control over the output of DVI processors.

Notes from the Secretary

Over the last year and a half the TUG DVI *Driver Standards Committee has been developing this document, the level-0 standard draft, which is presented now to the membership-at-large. The Committee plans to transform this draft into an official TUG standard as quickly as possible.*

If you have any comments on this draft, you may address them within two months after publication to the Secretary. Please note that it is intentional that this draft does not define inclusion of graphics, \special *syntax, page selection, etc. Refer to the section below and to the report published in TUGboat 12, no. 2 (May 1991) for the reasons.*

Owing to their length we have omitted the appendices from this publication. They contain the description of relevant file formats: Appendix A: DVI*; Appendix B:* GF*; Appendix C:* PK*; and Appendix D:* TFM*. These appendices are available over the Net, and may also be ordered on demand from the Secretary (the address is given at the end of this article).*

This Notes section is not part of the draft.

1 Purpose of the level-0 standard

The level-0 standard (hereafter called the *standard*) is meant to be a base standard to which all DVI-processing programs must adhere. It provides a base level of support for both DVI-to-output-device translators (so called *drivers*) and DVI-to-DVI preprocessors (e.g., dviselect). The standard hereafter calls such DVI-processing programs "DVI *processors*" or just "*processors*". This standard allows all reasonable documents to be rendered (i.e., printed or displayed) accurately. When we refer to accurate rendering, we mean that when the data generated by the DVI processor(s) are transmitted to an output device the latter shall produce a page accurately depicting the page described by the DVI file (disregarding resolution effects and output technology).

The basis for many of the specifications in this standard is the possible output of TEX82 although some requirements are based on assumptions that cannot occur with TEX82-based output; functions which can be implemented via a pre-processor are generally omitted (e.g., page selection and sorting).

2 The DVI file

DVI processors must be able to read and *interpret* any valid DVI file as specified in appendix A. They shall also correctly *render* any DVI file which falls within all of the limits specified below. If these requirements cannot be met due to limitations of the computer or the output device they shall be fulfilled as completely as possible and the limitations documented. Aside from this exception, these specifications are a *minimum*; good processors will probably be able to handle DVI files exceeding these limits (DVI files which exceed the limits are likely to be rare, but might still occur).

Explanation: This exception is necessary because certain output devices have varying capacity depending on the amount of on-board memory or similar conditions. For example, an HP LaserJet Plus with 512 KB of memory is capable of holding in memory only 3056 distinct downloaded characters; a full page bitmap is also not possible with this configuration.

DVI commands The DVI processor must be able to interpret every DVI command listed in Appendix A.

Explanation: Some commands, e.g., *put4*, are generally used for conditions outside those enumerated below; despite this, DVI-translating programs are expected to accurately interpret these commands and execute them if they do specify an action within the specified minimum limits.

Characters

Number of characters in a font

The DVI processor must be able to handle fonts which have characters at any code c in the range $0 \leq c < 256$.

Explanation: Some printers with download possibilities require fonts with more than a certain number of characters to be broken into two or more device fonts when downloaded to the printer. Please note that this requirement alone is not sufficient to allow the exception for device limitations given in section 2 to apply.

Character size

The DVI processor must be able to render any character up to a size of 600 pt (horizontal) by 800 pt (vertical) unless this is not possible due to device constraints as outlined in section 2.

Explanation: This size is the glyph size, not the size given in the TFM files. These two sizes are not connected; i.e., the glyph might be outside the bounding box given by the dimensions of the TFM files.

Number of characters per page

The DVI processor must be able to render a page containing up to 20 000 characters unless this is not possible due to device constraints as outlined in section 2.

Unusual characters

The DVI processor must correctly render any character which meets the specifications in appendices A, C, and B (if the processor uses GF format). This includes, but is not limited to: (a) characters with empty bitmaps (e.g., the SLiTEX fonts), including characters whose horizontal escapement is 0, (b) characters whose printable image is wider than their horizontal escapement, and (c) characters with a negative horizontal escapement.

Rules

Rule size

The DVI processor must be able to render rules of any size up to 600 pt (horizontal) by 800 pt (vertical) unless this is not possible due to device constraints as outlined in section 2.

Placement of rules on the page

The lower left corner of a rule is to be placed on the page at the location given by rounding the current DVI coordinates as indicated in section 2. The height and width of the rule are given by the formula $\lceil Kn \rceil$ where n is the dimension in DVI units and K is a constant which converts from DVI units to device units.

Explanation: Devices with aspect ratios unequal to one will need to maintain separate constants for vertical and horizontal dimensions.

No rule is rendered if $n \leq 0$, as specified in appendix A.

Number of rules per page

The DVI processor must be able to render a page containing up to 1000 rules unless this is not possible due to device constraints as outlined in section 2.

Stack The DVI processor must be able to handle DVI files whose *push/pop* stack nests up to 100 levels.

Positioning on the page

Location of the origin

The DVI processor must locate the origin $(0, 0)$ at a point one inch (25.4 mm) from the top of the page and one inch (25.4 mm) from the left side of the page.

Explanation: While these default margins are inconvenient for users of non-U.S.-sized paper, the advantage of having a universally standard default location of $(0, 0)$ and the widespread assumption of these defaults in most macro packages outweighs the inconveniences. For some DVI processors (e.g., screen previewers), this specification refers to a virtual page and not the physical output.

Changes in position due to characters and rules

The definition of DVI files refers to six registers, (h, v, w, x, y, z), which hold integer values in DVI units. In practice, we also need registers *hh* and *vv*, the pixel counterparts of h and v, since it is not always true that $hh = \text{pixel\_round}(h)$ or $vv = \text{pixel\_round}(v)$ where $\text{pixel\_round}(n)$ is defined as $\text{sign}(Kn) \cdot \lfloor \text{abs}(Kn) + 0.5 \rfloor$ with $\text{sign}(i)$ resulting in -1 if $i < 0$ and in 1 otherwise.

Whenever the DVI processor encounters an instruction that changes the current position, it must update h and v. If the change in position is due to a command which sets a character, the processor adds the horizontal escapement value from the PK or GF file to *hh* to get the new value for *hh*.

For a horizontal movement of x DVI units from any other command, *hh* will be set to $hh + \text{pixel\_round}(x)$ if $x < \textit{word\_space}$ for a horizontal movement to the right or if $x > -\textit{back\_space}$ for a horizontal movement to the left. *word_space* is defined as *space* − *space_shrink*, and *back_space* is defined as 0.9 *quad* if the processor uses TFM files. If the processor does not use TFM files the design size of the current font in the DVI file (after all necessary magnifications have been applied) may be used for a *quad*, and 0.2 *quad* must be used for *word_space*. If x exceeds the bounds outlined above, *hh* is set to $\text{pixel\_round}(h + x)$. In this way, rounding errors are absorbed by interword spaces.

For a vertical movement of y DVI units, *vv* is set similarly except that *vv* is set to *vv* +

pixel_round(y) if $-0.8quad < y < 0.8quad$ and set to pixel_round($v + y$) otherwise. This allows vertical rounding errors to be absorbed in the interline spacing while still allowing fractions and super- and subscripts to be printed consistently.

After any horizontal movement, a final check is made as to whether *dist* > *max_drift* with *dist* defined as abs(hh − pixel_round(Kh)) and *max_drift* defined as outlined below. If it is, then hh is set to pixel_round(Kh) + sign(*dist*) · *max_drift*. A similar check is made with vv and v. *max_drift* should be set to 2 for output devices with device units smaller than or equal to 0.005 in (0.127 mm), 1 for output devices with device units greater than 0.005 in (0.127 mm) but less than or equal to 0.01 in (0.254 mm) and 0 for output devices with device units greater than 0.01 in (0.254 mm).

Explanation: This method for tracking the positions is oriented towards the typesetting of text. It does not fix positioning problems with lines consisting completely of characters of a fixed-width font, where one line consists only of characters without any movements and the next line contains movements. Other problematic areas are line graphics produced with line segment characters in fonts. These line segments may not align.

Range of movement

The DVI processor must handle movements in the DVI file up to a total of $2^{31} - 1$ DVI units in any direction from the origin.

Objects off the page

Any printable object which would lie entirely off the physical page must not be rendered; any changes to positioning must still be obeyed. Any printable object which would lie partially off the physical page must either be clipped so that the portion of the object that lies off the page is not printed or else omitted entirely, unless this is not possible due to device constraints as outlined in section 2.

Explanation: Because some output devices do unpredictable things when objects are rendered partially or completely off the edge of the page, it is up to the DVI processor to make sure that objects printed partially off the page are handled correctly.

Fonts

Font numbers

The DVI processor must be able to accept a font number k, given by a *fnt_def* command, in the range $0 \le k < 256$.

Distinct fonts

The DVI processor must be able to handle any document containing 64 or fewer distinct fonts.

Specials A "special" is the parameter to the DVI commands *xxx1*, *xxx2*, *xxx3*, and *xxx4*. This standard does not define the meaning of any special. Specials not officially defined by the DVI processor standards committee should be flagged with a warning when read from the DVI file. If any specials are encountered that are ignored by the processor, the processor must issue a warning message. These warning messages may optionally be turned off at run time.

3 Configuration

It must be possible for the installer of a DVI processor to configure such things as the location and naming scheme of fonts, default paper size, etc., without having to recompile or relink the processor.

Explanation: "etc." means "make as many things configurable as possible." This should be more detailed (hint due to Karl Berry).

4 Font files

Font formats The DVI processor must be able to read PK fonts with the location specifiable at run time. The PK format is given in appendix C. GF support is optional. The GF format is given in appendix B.

Explanation: The PK format is the preferred format for bitmap fonts because (a) it is the most compact format in the TEX world and (b) included in the PK format are pieces of information about the font (e.g., the horizontal escapement in pixels for each character) which are essential for fulfilling the typesetting requirements of section 2.

The scaling number The magnification and resolution of a font are combined into a scaling number in one of two ways:

Resolution number
: The resolution number is given by *resolution* × *magnification* where the resolution is given in dots per inch (on devices with a aspect ratio unequal to one, the horizontal resolution should be used) and a magnification of 1 indicates normal sizing. This is the preferred specification for GF and PK files.

Magnification number
: The magnification number is given by 5 × *resolution* × *magnification* where both values are as above.

Magnifications

Minimum set of magnifications

The DVI processor must be able to use fonts at least at the following magnifications of its target resolution:

1 (magstep0),
1.095 (magstep0.5),
1.2 (magstep1),
1.44 (magstep2),
1.728 (magstep3),
2.074 (magstep4),
2.488 (magstep5),
2.986 (magstep6),
3.583 (magstep7),
4.3 (magstep8), and
5.160 (magstep9).

Explanation: The term magstep$n$ stems from the TEX and METAFONT control sequences with the same name. Its meaning is 1.2^n.

DVI processor authors are encouraged to support all possible magnifications.

Margin of error

If a DVI file requests a font at a size that does not exist, but the requested size is within 0.2 % of a supported magnification with the font at that size existing, the DVI processor must use the latter font without warning.

Explanation: TEX and METAFONT compute font magnifications with different precisions. Further, calculations done by TEX and/or a DVI processor are subject to roundoff errors. The margin prescribed is sufficient for accomodating most of these errors. It is *not* intended to compensate for fonts requested at an incorrect size.

Missing fonts If a font is missing the DVI processor must continue processing and, after issuing an appropriate warning message, deal with the missing font in one of three ways:

1. Insert appropriate white space where characters of the font would appear.
2. Insert black rectangles of the size of the characters given in the TFM file for the font.
3. Print the characters from that font at a different size or from another font at the same size.

If method 1 or 2 is used and the processor is unable to determine size information for the font in question, then the processor may simply ignore any character setting command that occurs while the current font is that font.

Under no circumstances should a missing font cause a fatal error.

◇ The TUG DVI Driver Standards Committee
c/o Joachim Schrod, Secretary
Technical University of Darmstadt
Institut für Theoretische Informatik
Alexanderstr. 10
W-6100 Darmstadt
Federal Republic of Germany
schrod@iti.informatik.th-darmstadt.de

Resources

New books on TEX

Victor Eijkhout

Editor's note: [Like the rest of this issue of *TUGboat*, this review was postponed from autumn 1991; by the time you read this, a corrected reprint of the subject book should be in circulation, incorporating some of the comments that appear here as well as other amendments.]

Even though English seems to be understood by just about everyone nowadays, TEX books in other languages still serve a useful purpose. Sometimes it looks as if the whole of Germany learned LATEX from Helmut Kopka instead of from Leslie Lamport, and in France Raymond Seroul's *Le petit livre de TEX* is very popular. In both cases, the rest of the world is getting a chance to see what it's been missing. Kopka's introductory volume is being translated, and Seroul's book has just appeared, under joint authorship with its translator, Silvio Levy.

A Beginner's Book of TEX (Springer Verlag, New York, 1991, ISBN 0-387-97562-4) is more than just a translation of the earlier book[1]. Levy is described as 'translator-turned-coauthor', and the most visible difference is the incorporation of the features of TEX version 3. The result is a rather handsome volume. For one, the text is very well-written, never feeling like a translation. The worst errors that I found were the misspelling 'wierd' which appears twice; the idiom 'head over heels' is used where something

[1] In this reviewer's opinion, however, the title has suffered from the translation. The original title had more of a *je ne sais quoi.*

like 'topsy-turvy' was meant, and the reader is told once that by finding an error in TEX 'you'll earn your prize and a place in the official listing of TEX's (former) bugs'. In general, the style of writing is the type of 'dialogue with the reader' that characterizes *The TEXbook*.

Another good point about the book is the rather open layout. The typefaces used are Times Roman and (its inevitable companion) Helvetica. Choosing these typefaces instead of Computer Modern, while in itself not too adventurous, removes the book immediately from the spheres of 'yet another book done with the TEX font'. The Computer Modern family is used to show examples of TEX output. A nice idea, although the effect is sometimes rather subtle, if just a single word of Computer Modern appears in a paragraph of Times.

My only criticism of the layout is that the book itself uses `\parindent=0pt`, so the output of some of the examples is different from what the ordinary user (who sticks to the default value of the indentation) will get. The authors should have made a remark about this, or prevented it from happening altogether.

The structure of the book is as follows. Chapter 1 is an introduction, chapter 13 is the 'Dictionary and Index', and in between are chapters that each treat an aspect of TEX, for instance modes, glue, paragraphs, math, or TEX programming. Although the final chapter is at 90 pages a generous one, and, well-stocked with examples, more than a mere index, I was most impressed with the expository chapters. They are meant for careful reading through them, rather than for easy reference (although the index refers back to them), but they contain an amount of information that is very respectable for an introductory book. It pleased me particularly to read the section on modes, a subject that is shunned by all other introductory books on TEX so far. The book contains many examples that illustrate their point well.

Of course, this book doesn't treat everything about TEX. The chapter on page layout has many examples, but, understandably, doesn't go very deeply into output routines. The control sequence `\expandafter` appears only in the Dictionary, and even there the reader is told that 'this subtle primitive is not for beginners'.

I have one comment about the Index/Dictionary, and that is that it contains too many irrelevant entries for my taste. It was the authors' idea to make the index refer to the examples 'by content', but it irritates me finding the likes of Humpty Dumpty and Bilbo Baggins all over the place.

In general, however, I found little to complain about in this book. There are hardly any TEX errors, and the ones that I found are not very serious. The worst error was that the authors claim that the keywords `height`, `depth`, and `width` have to appear in that order, whereas they may appear in any order. A case of misleading information is that the authors repeatedly recommend `\vglue` where the plain format of TEX version 3 has `\topglue`. Some other comments: the authors talk about 'the family `\fam1`' as if it were an identifier like 'the font `\MyFont`', whereas it is an assignment; calling `$` with category 12 an 'active character' because it prints as a dollar (page 173) is an unfortunate choice of words; and the reason that there are 18 `mu` to a quad may be obscure, but it is not 'only known to Knuth' as the authors state: the division of a quad in 18 basic units has been the standard for Monotype equipment for ages (this fact also appears in the space of the Computer Modern fonts: for the roman font the space is $1/3$`em` plus $1/6$`em` minus $1/9$`em`).

All of this is minor squabbling. This book does an admirable job of bringing together in single chapters enough information about topics in TEX for a starting TEXer to be able to 'typeset just about any document'. It is superb as an introductory reading text, and the Dictionary/Index can be used for reference later on.

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

New books on LATEX

Nico Poppelier

LATEX – Eine Einführung, Helmut Kopka, 3rd revised edition, Addison Wesley (Germany) 1991. hardbound, 375 pages.

LATEX – Erweiterungsmöglichkeiten mit einer Einführung in METAFONT, Helmut Kopka, 2nd revised edition, Addison Wesley (Germany) 1991. hardbound, 552 pages.

In view of the tremendous popularity of LATEX within the European TEX communities it is not really a surprise that two good books on LATEX, *LATEX*

– *Eine Einführung* (An introduction) and *LaTeX – Erweiterungsmöglichkeiten mit einer Einführung in METAFONT* (Possibilities for extension and an introduction to METAFONT), have appeared in Europe. These two books were written by DANTE member Helmut Kopka, and will soon appear in an English version.

The first book consists of nine chapters and four appendices. The chapters discuss the fundamentals of LaTeX, commands and environments, document styles and page styles, text producing commands, mathematical formulas (a very nice chapter, with lots of examples), pictures, user-defined structures and error handling. The appendices treat production of letters, BibTeX, fonts, and LaTeX extensions for the German language.

A description of MakeIndex and/or other indexing tools is missing, which is unfortunate since no LaTeX system is complete without BibTeX and MakeIndex. What is also lacking in this introductory volume is a description of the basic idea behind LaTeX, namely the separation of logical structure from visual structure. The author presents LaTeX as a bag of tricks. Maybe a big bag, and maybe useful tricks, but a bag of tricks nevertheless.

In chapter 4 the author discusses typefaces, typeface sizes and environments that explicitly concern layout, e.g. `center` and `flushleft`, before the ones that concern structure, such as `itemize` and `enumerate`. A tell-tale sign of the author's approach to LaTeX is the fact that already in chapter 3 he discusses various layout commands, such as line breaks and page dimensions.

The book contains a few errors, of which I give one example: on page 158 the author explains how to typeset chemical formulas. He explains how you can get all subscripts on one level, whether or not a superscript is present, by changing `\fontdimen16` and `\fontdimen17`. What the author forgets is that these are global changes, no many how many braces you put around them.

Nevertheless, it is a very useful book, full of information, with a list of all commands with a short description of each, and an extensive index.

The second book consists of three parts and three appendices. Part 1 describes extensions to LaTeX, for example the option 'german', $\mathcal{AMS}$-TeX and its fonts, SliTeX, PiCTeX and MakeIndex. Since MakeIndex is not an extension, but an essential part of LaTeX, it should have appeared in the first book. Furthermore, the description of $\mathcal{AMS}$-TeX should be replaced by a description of $\mathcal{AMS}$-LaTeX.

Part 2 is called 'LaTeX for advanced users'. It discusses the structure of the LaTeX system, including document styles, and gives a short introduction to TeX. Part 3 is the short introduction to METAFONT. The appendices treat WEB, the various other programs related to TeX, and a sample application in CWEB.

All in all, this is a useful book, because it brings together information about the complete TeX system. Also, it is the only real description of LaTeX's document-style mechanism.

The author has told me that the English version of his books will not be a mere translation of the German originals, but an internationalized version. I would like to suggest that in the rewriting process parts specific to the German language are removed in favour of international extensions to LaTeX, e.g. 'a4' and 'babel', that some parts are moved from book 1 to book 2 or vice versa, and that book 2 discusses the new font-selection scheme for LaTeX.

A general criticism: these books appear to be produced from low-resolution camera copy, which is a practice I think should be abandoned. Some of the illustrations in the introduction to METAFONT, in the second book, are particularly bad. TeX's high-quality output deserves better, namely

- a *real layout*, i.e. not one of the standard styles modified by the author or the publisher, but one created by a *real* designer
- *real* typefaces, i.e. not Computer Modern at a low resolution. Computer Modern is not synonymous with TeX! Besides, it's old-fashioned.

Finally: let's hope that these books appear soon, so that they can still be used for a number of years. After all, LaTeX 3 is in the making ...

◇ Nico Poppelier
Elsevier Science Publishers BV
Amsterdam, The Netherlands
n.poppelier@elsevier.nl

Q & A

JUST PLAIN Q&A: Of Partitioned Matrices and Doublespacing.

Alan Hoenig

This column serves as a forum in which people can seek answers to TEX questions, with an emphasis on plain TEX. Questions at *all* levels of difficulty are welcome. We hope to hear from you.

How Do You Typeset Partitioned Matrices?

David Handelman sent a request for aid on the typesetting of partitioned matrices. His question lay on my desk for a shamefully long time, and my apologies to him. I hope he will agree that this response is worth the wait, as a superior set of tools has appeared to make his life much easier.

David seeks to typeset partitioned matrices, which are things of this ilk:

(A) $$\left(\begin{array}{c|c} A_1 & Z \\ \hline A_3 & A_4 \end{array}\right);$$

that is, rectangular arrays of expressions (matrices) pierced through by solid or dashed vertical and horizontal line segments.

Since David posed his query, Mike Spivak has created the LAMS-TEX macro package which handles this type of thing. At the July 1991 TUG meeting in Dedham, Mike announced the placing of this extensive package in the public domain. It can be downloaded from any of several TEX archives.

Here's how you can typeset matrix (A). First, load in the appropriate macro and style files at the beginning of the document.

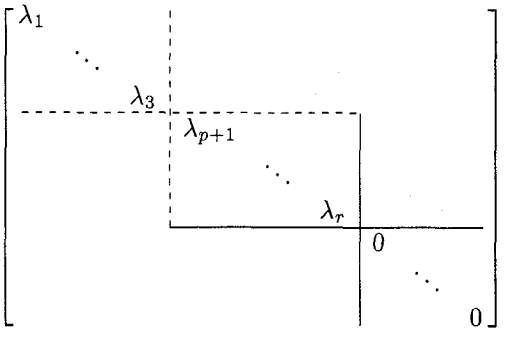
```
\input /lamstex/amstexl
\input /lamstex/lamstex
\input /lamstex/ptmatrix
```

File `amstexl` is subset of normal AMS-TEX, and `lamstex` is the core set of LAMS-TEX macros. The partitioned matrix macros are in `ptmatrix`. In math display mode, enter

```
\left(
\partition
\matrix
A_1&Z\\
A_3&A_4
\endmatrix
\vsolid 1:02
\hsolid 1:02
\endpartition
\right)
```

where the syntax resembles that of AMS-TEX with a few obvious additions. The `\vsolid` command specifies a vertical solid rule after column one extending from the top of row 0 (bottom of the matrix) to the top of row 2. `\hsolid` directs the creation of a solid horizontal line after row 1 which extends from the zeroth column (left of the matrix) to the second column.

More complicated examples are possible, such



which comes from

```
\left[\partition
\matrix
\lambda_1\\
&\ddots\\
&&\lambda_3\\
&&&\lambda_{p+1}\\
&&&&\ddots\\
&&&&&\lambda_r\\
&&&&&&0\\
&&&&&&&\ddots\\
&&&&&&&&0
\endmatrix
\vdashed 3:39
\vsolid 6:06
\hsolid 3:39
\hdashed 6:06
\endpartition\right]
```

Many additional options exist for twiddling with the position and appearance of the rules.

The LAMS-TEX macro package contains many powerful and useful features. It's an extension of AMS-TEX with the functionality of LATEX (but more concise syntax). Automatic numbering schemes are very flexible, and can easily be modified for special circumstances. Extensive table-making abilities are part of the package. It's possible too to create complicated and professional commutative diagrams along with partitioned matrices. An index program comes with LAMS-TEX, and LAMS-TEX

now interfaces with BibTeX. No commands are fragile. And much, much more. You may contact the author via e-mail at spivak@math.rice.edu. Manuals for the package may be purchased from the TeXplorators Corporation, 3701 W. Alabama, Suite 450-273, Houston, TX 77027.

Controlling Interline Spacing in TeX and LaTeX

I'd like to present a limited discussion of doublespacing on behalf of the many people over the years who have wanted to doublespace their TeX and LaTeX documents. At first blush, you might wonder why such an anachronism is needed in this day and age, but copy editors still demand a doublespaced manuscript to ensure enough room for their red pencils. If this is why you need doublespacing, then it's reasonably easy to jury-rig important doublespacing details. If you need doublespacing because of the archaic needs of a thesis style, say, then you can embed all the proper doublespace formatting in your style file. (Or perhaps the style file you use already takes it into account.)

It's not enough simply to reset \baselineskip, because there are plenty of situations where other parameters such as \lineskip control the interline spacing. A better idea is to reset all the relevant parameters by means of the \openup command:

```
\openup1pc
```

or

```
\openup \baselineskip
```

for example.

Even this is not enough, though. There may well be groups within which \offinterlineskip has been set and spacing is controlled by \strut's. (This is often the case within specialized table macros.) Here is one way to extend a strut after you've \openup'ed the baseline.

```
\newbox\newstrutbox
\setbox\newstrutbox=
 \hbox{\vrule height.7\baselineskip
  depth .3\baselineskip width0pt}
\setbox\strutbox=\box\newstrutbox
```

It's easy to combine all these details into a single macro.

You may also want to increase the space above and below a display with commands like

```
\advance \abovedisplayskip by 6pt
```

or whatever, and similarly for \abovedisplayshortskip, \belowdisplayskip, and \belowdisplayshortskip.

But of course this will not give exact doublespacing in all circumstances. If you \openup1pc, then a footnote whose original baseline is 8 pt will have a new baseline of 20 pt (remember, 20 = 8+12) rather than the 16 pt that you might prefer. If you are running off a quick draft for a copy editor, you probably don't care. If this is your thesis, well, you'll have to work a bit harder with your \footnote macro.

For LaTeX users, the same considerations apply, but you must implement them in proper LaTeX syntax. The details are in two files that are easily ftp'able. The first, put together by S. Page and subsequently modified by J.-F. Lamy, is doublespace.sty at sun.soe.clarkson.edu in the directory pub/tex/latex-style. The same file, with additional modifications by S. Rahtz to render it useful when using the new font selection scheme of Mittelbach and Schöpf, resides in the ymir archive in [anonymous.tex.inputs.latex-contrib]. Check the TUG resource directory for assistance in retrieving these (or any) files from the archives.

Department of Amplification

Bernd Raichle, Dante coordinator for german.sty, has pointed out that the macros I presented last time for playing around with \fontdimen parameters could be easily defeated by users with a sense of the macabre. (But my apologies nevertheless for not being more rigorous in my own testing.) He has taken the trouble to fortify those macros, for which I thank him, and I present them here with his comments, together with a short test.

```
\font\roman=cmr10
\font\specroman=cmr10
\roman
\newdimen\savedvalue
 \savedvalue=\fontdimen2\roman
\newdimen\specialvalue
 \specialvalue=19.99pt
\newif\ifreset
 % changed global, indicates need to reset
\newif\ifspec
 % changed local, indicates current spacing
% For normal spacing,
% call \selectspacing with \specfalse
% and switch to \roman font
%
\def\rm{\specfalse \selectspacing \roman}
% For large spacing, call \selectspacing
% with \spectrue, remember that we have
% to change the spacing after the group
% and switch to \specroman font.
%
```

```
\def\specrm{\spectrue \selectspacing
   \aftergroup\selectspacing \specroman}
% Switch to large spacing and remember
% in \ifreset that we have to switch
% back after the group.
%
\def\setdimen{%
  \fontdimen2\specroman=\specialvalue
  \global\resettrue}
% Switch to normal spacing.
% If there is a call to
% \selectspacing after the group,
% there's no need to switch.
%
\def\resetdimen{%
 \fontdimen2\specroman=\savedvalue
 \global\resetfalse}
% This macro does two things:
%  1. If we have changed to larger spacing,
%     we switch back to normal spacing
%     (only if \resettrue).
%  2. If \ifspec is true for the
%     current group we switch to
%     larger spacing.  (The correct \font
%     change to \specroman is done
%     by TeX if this macro is called
%     after a group.)
%
\def\selectspacing{%
  \ifreset \resetdimen \fi
  \ifspec \setdimen \fi}

% A short test:
%
\obeylines
\rm n o r m a l
\specrm s p e c
{\specrm s p e c
 \rm n o r m a l
 \rm n o r m a l
 {\specrm s p e c}
 n o r m a l
 \specrm s p e c
}
s p e c
\specrm s p e c
\rm n o r m a l
```

◇ Alan Hoenig
17 Bay Avenue
Huntington, NY 11743
(516) 385–0736
ajhjj@cunyvm

Tutorials

Elementary Text Processing and Parsing in TEX

— the appreciation of tokens —

L. Siebenmann

Background

Token lists make up the material found in the upper digestive tract of TEX, and token list registers are very useful means to improve TEX's digestion. I begin this tutorial by showing how to do elementary 'text processing' with token lists. Then I apply this 'token list processing' to parsing of classical keyword syntax where the keys come in any order and their fields (or arguments) are terminated by nothing more than the next keyword. This processing and parsing are simple concepts that many TEXperts, not to mention beginners, have largely neglected. I find that TEX assimilates them well, and hope they will see wider use in the future.

I originally explored this parsing as a possible method to fix a subtle line-breaking bug in $\mathcal{AMS}$-TEX bibliographies that was pointed out by Barbara Beeton in 1990. This remains a convenient example to test methods; but in truth an academic one, since Michael Downes [Do] has successfully fixed the bug (for version 2.1 of July 1991) using a very different `\vbox` trick proposed by Don Knuth. The general subject of parsing in TEX language, to which this tutorial contributes two methods called (A) and (B) below, was introduced by W. Appelt in his book [App].

I want to thank Michael Downes, Victor Eijkhout, and Ron Whitney for contributing many helpful comments as this tutorial evolved. My ignorance and uncertainty about what all can or cannot be found in *The TEXbook* was a problem that delayed this tutorial; one remedy I enjoyed using is surely of interest to readers of *TUGboat*, namely string searches in an online version of *The TEXbook*.[1] Perhaps a "HyperTEX" soon will combine this brute force information processing with *The TEXbook*'s beauty and readability. It will

1) The `.tex` file for *The TEXbook* can for example be obtained by anonymous ftp from the archives

`labrea.stanford.edu`
`rusinfo.rus.uni-stuttgart.de`

It fits on a diskette and can conveniently be used on a microcomputer.

not be long before the mass of articles in *TUGboat* merits similar treatment.

Section 0. Token Lists and Registers

As TEX reads in a file, it builds,[2] from the incoming stream of characters (or octets), a closely corresponding stream of 'tokens', i.e., of control sequences and characters-with-category. For example, the ASCII characters `\TeX` — including spaces after `X` — become a single control sequence token representing the TEX logo, and an ordinary (English) word becomes its usual sequence of ASCII characters each with category 11 (= letter). The details (worth re-reading often!) are found in *The TEXbook*, particularly [Chapter 7].[3]

For our purposes, it is not too far from the truth to say that a control sequence is a token that one can specify in the input stream using a backslash followed by a finite sequence of letters (category 11) or a backslash followed by a single character of another category. However, once inside TEX, this control sequence name is, for efficiency, left in a cloakroom, and, in all internal activities, it is represented by a number of fixed length (four or five octets). This means that a control sequence with a long name is no harder for TEX to manipulate than one with a short name.

Control sequences come in many formally recognized varieties, somewhat like the professions of man. The command `\show\mycs` should make TEX tell you the 'profession' of `\mycs` along with some further details: perhaps `\mycs` is a macro, a token list register, a dimension register, a primitive, undefined, etc. We are most concerned with *macros* and *token list registers*. Both of these are 'white-collar workers' that would never get down to the dirty details of typesetting without help from typographic 'primitives' like `\char` and `\hbox`. Both have the same sort of information content, namely a token list, which means they are in some sense just containers holding other tokens! What makes macros and token lists different is their syntax and activity; for example, macros naturally expand while token list registers are fairly inert.

Let us get down to specifics. Given, for example, the token list produced by Plain TEX reading `{\TeX} is useful`, as every TEX programmer knows, we can define a macro called `\mymacro` whose content or 'expansion' is this token list, by typing

```
\def\mymacro{{\TeX} is useful}
```

Check this by executing `\show\mymacro`; there are eight alphabetical characters, two space tokens, one control sequence `\TeX`, and two brace characters.

But, we can also allocate a token list register `\mytoks` by typing `\newtoks\mytoks` and give it the same contents by typing

```
\mytoks={{\TeX} is useful}
```

Here the equal sign is optional; we will often omit it. One checks the contents by executing `\showthe\mytoks`.

There are exactly 256 token list registers `\toks0`,...,`\toks255` and `\mytoks` has been made to stand for one of these by use of a primitive `\toksdef` which is called by the macro `\newtoks` above. This limited number of registers is fixed by the structure and documentation of TEX, whereas the number of control sequences (= hash size) is either flexible or decided by the programmer who compiled your TEX. OzTEX for example has a configuration file letting you set hash size (up to 6500) along with many other parameters.

There is a clear distinction between a token list register and the token list it contains — analogous to the distinction between the wine bottle and the wine. Thus it is an 'abuse' of language (in the benign sense of N. Bourbaki) when one nevertheless talks of 'a token list `\mytoks`'. The word 'toks' will often be used in what follows as an informal abbreviation for 'token list'.

The contents of `\mymacro` can be transferred to `\mytoks` and the other way around as (1) and (2) indicate.

```
\mytoks=\expandafter{\mymacro}                (1)
\expandafter\def\expandafter\mymacro
  \expandafter{\the\mytoks}                   (2)
```

To understand these formulas, recall that the primitive `\expandafter` serves to modify TEX's reasonably 'straight-ahead' expansion procedure by expanding the token next-but-one to the right. Thus, in (1), it causes `\mymacro` to be replaced by its expansion token list before the token list register `\mytoks` has its value assigned. In (2), the first `\expandafter` acts on the second which then acts on the third which acts on `\the` to replace `\the\mytoks` by the token list in `\mytoks` to give the intermediate result

```
\def\mymacro{⟨the toks in \mytoks⟩}
```

2) With its 'lips', to use Knuth's helpful digestive tract analogy. Token list manipulation is done in TEX's 'mouth' and so could be called 'mastication'.

3) In the absence of more explicit indications, citations in square brackets refer to *The TEXbook*.

Further uses of \expandafter will occur below.

Try now the following less well known alternative to formula (2):

```
\edef\mymacro{\the\mytoks}          (2*)
```

An alert reader may wish to protest at this point that this formula will fail whenever the token lists in \mytoks would itself admit expansion by \edef. Wrongly! In fact, although \edef *usually* does a maximum of the 'formal' expansions, it does just a single expansion of anything of the form \the⟨*token register*⟩; see [p. 216 (top)]—a very convenient exception.

Speed as well as elegance argues for using formula (2*) rather than (2). I was surprised to find that (2*) runs at over twice the speed of (2) or of (1). (In principle, speed ratios could vary with the implementation of TEX.)

It is probably because of this 'material equivalence' of macros (simple ones without parameters) and token list registers, that most TEX users and programmers very much neglect token list registers. Notwithstanding, I hope to gradually convince the reader that token list registers are helpful, both conceptually and practically, and deserve a place on every TEXpert's workbench.

Some pitfalls involving token lists

> *Exposition in physics should be as simple as possible. But not simpler.*
>
> *A. Einstein*

(1) Where token list registers are concerned, we should always restrict ourselves to token lists that are *balanced* in the usual sense that the grouping symbols { and } balance. For example {} and {{}{}} are balanced while }{ and }{ are not. Knuth assures us [p. 375 (bottom)] that it is impossible to put an unbalanced token list into a token register.

Note that there is absolutely no requirement that a token list in a toks register be balanced with respect to other standard grouping pairs such as \bgroup, \egroup and \begingroup, \endgroup.

(2) Be prepared for some mind-boggling distinctions among the three grouping pairs just met. For example, in the token assignment \mytoks{...}, the { can be replaced by \bgroup but not by \begingroup. On the other hand } cannot be replaced at all! This is carefully documented on page 276 of *The TEXbook*.

(3) To put one sharp character #, with its usual category (6=Parameter), into the token list that is the expansion text of a macro \mymacro requires one to input two sharps ##. Thus \def\mymacro{##} makes the expansion a single sharp. The single sharp in macro definition input is reserved for macro parameters. In token list register input, this complication does not exist: \mytoks={#} puts one sharp into \mytoks. Many (all?) output functions to screen or file double each (category 6) sharp, notably \show and \showthe; thus \mytoks={#}\showthe\mytoks yields ##. The reader will have to be aware of doubling phenomena for # to understand the formulas for parsing in the sidebar of section 2. See [pages 203–204, 216, 228].

(4) *About* \edef *and its cohorts.* Each macro has an expansion to a token list. It is tempting to believe that, analogously, (balanced) token lists have an 'immediate expansion' provided by \edef. To expand the token list in \mymacro execute

```
\edef\mymacro{\mymacro}
```

Use \show\mymacro before and after to see the effect; the expansion is in some sense complete and immediate.

Alas, this 'complete expansion' is not always defined, and when defined may be utter nonsense; for example, if the token expansion for \mymacro is \def\aaa{AAA} where \aaa is not already defined then TEX will balk, while if \aaa is defined to be aaa then one gets \def aaa{AAA}!

TEX also has a surprise in store for you if you believe that, when you change an occurrence of \def to \edef, the (unexpanded) definition text read in will necessarily be the same for each; see [Exercise 20.17].

The rules for \edef are carefully laid out in *The TEXbook* [p. 215 (bottom) and p. 216 (top)]. The double bends there are justified by the subtlety of \edef, not by its rarity or lack of importance! The rules are all the more worth learning because they apply with only minor modification to \mark{...}, \message{...}, \errmessage{...}, \special{...}, and \write{...}; see [p. 216 below 20.16]. Roughly speaking, \edef and these 'cohorts' do all the formal expansion that is possible subject to an overriding condition that this expansion process should change nothing in the TEX environment other than the ultimate expansion token list for the macro being defined. It in fact does slightly less than that because of the important single expansion rule [p. 216 (top)] for \the⟨*token register*⟩ that we have already encountered.

Always keep in mind that \edef and its cohorts can only be used when the programmer has such intimate knowledge of the toks to be expanded that he can guarantee the results are well-defined and

suitable for his purposes. (In other cases, simpler tools such as `\expandafter` and `\noexpand` may prove useful.) Since the `\edef` primitive is powerful, and can often do more for us in less time and with less programming effort than competing tools, its (prudent!) use is to be encouraged.

The single expansion rule above for `\the`⟨*token register*⟩ with respect to `\edef` and its cohorts offers the only way I know to efficiently suppress expansion of a long list of tokens; the primitive `\noexpand` applies to only a single token.

Section 1. Elementary 'text processing' with Token Lists

It is well known that TEX can dabble in computer graphics (LATEX does) and even in number theory [p. 218], so it should come as no surprise that it can master the rudiments of classical text processing. But although this ability is obviously relevant to TEX's main purpose, typesetting, it seems little attention has been paid to it.

The most basic operations of text processing on a list of characters (or more generally of tokens) are:

(a) *copying.*
(b) *concatenating* two lists x and y to form a composed list xy.
(c) *searching* for one list x in another z (is x a sublist of z?).
(d) *splitting* a token list z at a sublist x (known to be present) into parts a, x, and b, so that z is the concatenation axb.

The problems these token list processing operations pose for us are practical problems of coaxing TEX to perform these useful operations efficiently. It turns out that most of them are a bit tricky to define, but reasonably compact and efficient once defined. To keep the formulas simple, I often do not give the operations a catch-all syntax, as might be desirable in a large macro package. That can be left to the programmer.

One can at first imagine that the token lists are segments of English prose, but in general there are control sequence tokens as well as character tokens. The situation is somewhat analogous in computer printer scripts of the 1970's and in some wordprocessor files that represent changes of font style, etc., as tokens intermixed with the ordinary characters.

TEX forces on us a very stringent notion of equivalence for token lists, namely one-to-one order preserving correspondence of the tokens in the lists so that corresponding tokens are *identical* (not just `\let`-equal or identical-after-expansion). Coarser notions are probably best approached by doing some preliminary macro expansion. Assuming two toks are the expansions of `\mymacro` and `\thymacro` respectively, the standard test for equivalence uses `\ifx` as in

```
\ifx\mymacro\thymacro\message{EQUIVALENT}
 \else\message{INEQUIVALENT}\fi
```

We assume below that `\xtoks`, `\ytoks`, `\ztoks`, `\atoks`, `\btoks`, are allocated token list registers, cf. section 0.

Copying token lists

To copy the toks in register `\atoks` into the toks register `\btoks` is a simple matter:

```
\btoks=\atoks
```

This is analogous to `\let\b=\a`; speed is great and independent of the contents of the register `\atoks`. Quite the opposite can be said of the alternative formula `\btoks=\expandafter{\the\atoks}`.

There is another form of copying: macro arguments, written `#1`, `#2`, etc., represent token lists too and, in the definition of a macro with arguments [Chap. 20], they can be stuffed directly into a token list register or a macro expansion. See the splitting macro `\SPLITT@` below for a simple example.

The `\read` primitive provides still another form of copying: it reads in a line from an open file `\myfile` thus:

```
\read\myfile\mymacro
```

converting it to the expansion toks of the macro `\mymacro`. The inverse operation can be accomplished[1] by

```
\mytoks=\expandafter{\mymacro}
\immediate\write\myfile{\the\mytoks}
```

Recall that `\write` is one of the cohorts of `\edef`; this is another use of the 'single expansion' phenomenon. Beware that because of category codes

1) Ron Whitney [Wh] has shown how to do this inverse operation using `\meaning` in place of a toks register. His approach is preferable for non-immediate writes which are often used in index construction; the difficulty with the toks register approach is revealed by executing

```
\mytoks={aaa}\write\myfile{\the\mytoks}
\mytoks={bbb}\write\myfile{\the\mytoks}
```

Whitney's approach is much simpler and not less effective than an earlier one of Todd Allen [p. 377].

and TEX's reading conventions these two operations may not be strictly inverse one to the other.

Concatenating

We propose to concatenate \xtoks and \ytoks and put the result in \ztoks.

The following simple formula gives the right idea but fails dismally

```
\ztoks{\the\xtoks\the\ytoks}          (1x)
```

because of the distinction between wine bottle and wine. It is well known that cunning use of the primitive \expandafter can correct this. *We assume* \let\e=\expandafter *henceforth.* The most usual formula is impressive

```
\e\ztoks\e\e\e
  {\e\the\e\xtoks\the\ytoks}          (1a)
```

and also fun to expand: to begin, the five odd-numbered tokens from the left (all \expandafter's) go off in sequence like a long fuse and detonate the last \the to produce an intermediate form:

```
\ztoks\e{\the\xtoks⟨the toks in \ytoks⟩}
```

From this point, a short fuse consisting of just one \e similarly detonates the first \the to produce a second intermediate result

```
\ztoks{⟨the toks in \xtoks⟩%
  ⟨the toks in \ytoks⟩}
```

which is then normally executed to give the desired result.

Do not bother to memorize intimidating formulas like (1a)! You just have to remember the intermediate stages and work backwards stringing out your fuse lines of \e's.

And do not go out of your way to use them in serious programming! They often execute more slowly than than alternatives. In this case there is an alternative that *entirely* avoids \expandafter, exploiting \edef instead:

```
\edef\dummy{\ztoks={%
 \the\xtoks\the\ytoks}}\dummy          (1b)
```

It executes 15% faster than (1a). There are many less elegant solutions that execute as quickly, e.g.

```
\edef\dummy{\the\xtoks\the\ytoks}
\ztoks=\e{\dummy}
```

Concatenation can also be done directly for the toks of macro expansions; the trickery is much the same. Indeed, given \x and \y, we can define \z as follows

```
\e\e\e\def\e\e\e\z\e\e\e{\e\x\y}      (2a)
```

or by

```
\toks0=\e{\x} \toks2=\e{\y}
\edef\z{\the\toks0 \the\toks2}        (2b)
```

In (2b), we have used two of the five local 'scratch' toks registers, numbers 0, 2, 4, 6, 8, that TEX reserves for temporary storage [p. 346]; this merely avoids allocating special registers for the purpose, using \newtoks. Caution: Many technicalities arise in using explicit registers. For one, the odd registers 1, 3, 5, 7, 9 are reserved for global definitions; see [p. 346]. For another, space after the second \toks0 above is obligatory. Indeed, without it (or some alternative like \relax), TEX expands \the\toks2 in the process of assimilating \the\toks0 and then a full expansion of \the\toks2 is attempted, which is not what we want here.

Searching for one token list in another

Our goal is to decide whether a toks ⟨*toks sought*⟩ is equivalent to a sublist of another toks ⟨*toks to be searched*⟩.

The notion of a sublist of a (balanced!) token list that we shall use is restricted to balanced sublists occurring at nesting level zero for the TEX grouping symbols { and }. Such sublists of a balanced list z are precisely those sublists x inducing a splitting $z = axb$ with all three of a, x, and b balanced. Call such sublists *admissible*. For example, the sublist st in the seven token list r{st}uv is a balanced but *inadmissible* sublist, being at brace level 1. On the other hand, {st}u is a balanced and admissible sublist. (If this notion is not to your liking, see [p. 376 (middle)].)

The tool we use for searching is the full TEX macro mechanism including parameters and match text. As Knuth treats search macros in a highly condensed fashion in the dirty tricks chapter [Appendix D], a motivated discussion will be given here.

To get the main idea, observe that a definition

```
\def\mymacro#1⟨toks sought⟩{...}          (*)
```

of a macro with match text [p. 203] will make \mymacro look for the first occurrence of the token list ⟨*toks sought*⟩ in the input after \mymacro[2] and make #1 be the token list (possibly empty) between the two.

This approach imposes a significant restriction on ⟨*toks sought*⟩ that is admittedly quite undesirable. Since it is a macro match text, ⟨*toks sought*⟩ *must contain no brace characters*, for if it did TEX would see a shorter macro definition in (*)!

2) If there is none before the next occurrence of \par an error will result, unless \long\def replaces \def.

Next observe that to prevent trouble in case ⟨*toks sought*⟩ is absent, we can apply such a macro to, for example:

⟨*toks to be searched*⟩`\premarker`⟨*toks sought*⟩`%`
`\postmarker\endmarker` (**)

Now, of course, the ⟨*toks sought*⟩ is always found and the search problem is converted into a question of *where* it is found: is it in the ⟨*toks to be searched*⟩ or between markers? For this, one can apply to (**) a macro with more complicated match text — as follows:

```
\def\searchmacro#1⟨toks sought⟩%
   #2#3\endmarker{...}
```

(We have still to decide on the macro substitution text `{...}`!) What happens when this is applied to (**)? Because of `\endmarker` the macro uses up the full text (**), which is all to the good — a leftover could cause havoc. The argument `#2` will be the token immediately following the first occurrence of ⟨*toks sought*⟩ in (**) and we conclude that `#2` is `\postmarker` precisely if ⟨*toks sought*⟩ failed to occur in ⟨*toks to be searched*⟩. Thus after setting out preliminary material

```
\newif\iffound
\def\postmarker{\uniquecs}
```

we specify the substitution text `{...}` to be:

```
{\def\this{#2}\ifx\this\postmarker
  \foundtrue\else\foundfalse\fi}
```

Putting all this together we have a search macro `\searchmacro` for a fixed toks ⟨*toks sought*⟩.

Several improvements are given in the 'production version' (3) below:

(a) allow ⟨*toks sought*⟩ to vary; this requires a somewhat confusing layer of indirection.
(b) allow both ⟨*toks sought*⟩ and ⟨*toks to be searched*⟩ to be specified in terms of a token register or macro as well as by direct typing; the solution is to specify ⟨*toks to be searched*⟩ by anything whose first expansion is ⟨*toks to be searched*⟩, and similarly for ⟨*toks sought*⟩.
(c) make direct typing of ⟨*toks sought*⟩ and ⟨*toks to be searched*⟩ convenient (our first attempt ignores initial spaces); the strings `0` (zero, not 'oh') and `@@` in the production version permit this.
(d) keep the macro and related apparatus out of the way of non-programmers by use of `@` with category 11 (letter).

The production version below was adapted from one in $\mathcal{A}\mathcal{M}\mathcal{S}$-TEX by Mike Spivak, which in turn was adapted from [p. 375]. I had to generalize somewhat to allow `#1` to be a token list rather than a character and to assure features (a)–(d). Also I spent a few extra control sequences on readability.[3)]

Roughly speaking, the `\IN@0#1@#2@` below sets the condition `\ifIN@` to true if the toks for `#1` is a sublist of the toks for `#2` and otherwise sets it to false. More precisely `#1` and `#2` should be things whose first expansions are the toks in question — so that arguments `#1` and `#2` can be of the form `\mymacro` or `\the\mytoks`.

```
\newif\ifIN@
\def\IN@{\e\INN@\e}
\def\INN@0#1@#2@%
 {\def\NI@##1#1##2##3\ENDNI@
   {\ifx\m@rker##2\IN@false
   \else\IN@true\fi}%
  \e\NI@#2@@#1\m@rker\ENDNI@}
\def\m@rker{\m@@rker}                    (3)
```

There are some reasonable technical restrictions on this macro. It is to be defined and used inside macro packages where `@` has been given catcode 11 (= letter). Neither token list produced by `#1` and `#2` should contain a `\par`[4)], nor a character token `@` with catcode 11 — something easily avoided as they are either under the programmer's control or come from the user's world where `@` has catcode 12 (= other) or 13 (= active). Further, neither should contain a token (like `\m@rker`), whose expansion begins with `\m@@rker`.

There is also one annoying restriction explained above. *The toks for* `#1`*, i.e.,* ⟨*toks to find*⟩*, must contain no braces.*[5)] However, braces (balanced of course) are permitted in ⟨*toks to be searched*⟩.

The above production version is admittedly very technical; fortunately no understanding of how all the the details work together is essential for what follows. Incidentally, the splitting macro below is more transparent and could serve as a stepping stone.

---

3) For hints on recovering these examine [p. 375].
4) To allow `\par` one uses `\long\def` in place of `\def`.
5) One way to work around this restriction without resorting to the slow token-by-token approach of [p. 376 (middle)] might be to use the `\meaning` primitive to first convert braces to category 12 characters, cf. Ron Whitney's note [Wh]. This also gets around the blanket restriction to 'balanced' token list. However, it may require you to use `\write` and `\read` to reconstitute control sequence tokens from category 12 characters.

Splitting at a sublist

Suppose we know (from the test above, for example) that the toks (with no braces) in \xtoks is a sublist of the toks in \ztoks. Then we typically want to put into \atoks the segment of \ztoks up to the first occurrence of \xtoks and put into \btoks the segment following that occurrence of \xtoks. This is to be accomplished by the syntax

```
\SPLIT@0\xtoks @\ztoks @
\atoks=\Initialtoks@
\btoks=\Terminaltoks@
```

where \SPLIT@0#1@#2@ carries on the basic conventions and design features for \IN@0#1@#2@ set out above. The macro definitions required are

```
\newtoks\Initialtoks@
\newtoks\Terminaltoks@
\def\SPLIT@{\e\SPLITT@\e}
\def\SPLITT@0#1@#2@%
 {\def\TTILPS@##1#1##2@%
   {\Initialtoks@{##1}%
   \Terminaltoks@{##2}}%
  \e\TTILPS@#2@}                    (4)
```

We have now established basic processing functions for TEX's token lists that are generalizations of well known text processing functions, and that execute at a useful speed. They can be used to edit pieces of text before printing them, and more importantly to build new macros that provide users with syntax with flexible options. This second 'parsing' theme will be pursued in the next section.

I also recommend use of token list processing deep within macro packages; for hints on this sort of application I suggest reading about Knuth's list macros [Appendix D, p. 378–379] and Appelt's stack macros [App, Chap. 5]. Incidentally, TEX offers some *ready-made* text processing control sequences such as \uppercase and \lowercase.

Section 2.
From Text Processing to Keyword Parsing

One of the most powerful, convenient, and wide spread syntaxes one encounters on classical computers is the 'keyword option' system. W. Appelt [App] has advertised this system in TEX programming, and provided a practical sort of recipe to implement it, after a first simple example by Knuth [p. 376 (top)]. Here we will provide recipes offering improvements such as more general syntax, potentially greater speed or capacity, or more compact formulas. The most general recipe is the the second below, called (A); it will be simple application of our token list processing of section 1. But a more subtle process (B) will often give better results in case the keywords are macros.

An ad hoc parsing process

The keyword option system will be illustrated first by a \special command from Tom Rokicki's dvips postscript printer driver for TEX. His syntax summary is:

```
\Special{psfile="filename"[ key=value]*}   (1)
```

Here the possible keys are the words: hoffset, voffset, hsize, vsize, hscale, vscale, angle, and each of these keys calls for a suitable quantity in place of value. I have perversely written \Special for \special here so that (1) and (2) can soon be assigned another meaning.

A specific example is

```
\Special{psfile=myfile
  angle=90 hscale=50 vscale=50}          (2)
```

which prints the PostScript graphics file myfile rotated 90 degrees at scale 50 percent. The central point to note is that the user can specify *any number* (or zero) of keys in *any order* he pleases.

This command is interpreted by dvips (a printer driver) after a preliminary expansion by TEX.[1] But *let us imagine that we want TEX to interpret a control sequence with this sort of syntax.* For example, one might want a TEX macro \Special with *identical* syntax, that provides, in addition to what Rokicki's \special gives, a TEX box into which the printed graphics nicely fits. Of course, such a \Special will normally also appeal to \special after composing a suitable box.

How can TEX understand or 'parse' (2)?

By making \Special a one-argument macro, TEX can efficiently isolate the guts of (2), namely psfile=myfile ... vscale=50, and store it as a token list T in a token list register, say \Ttoks. This is the first (easy) step of parsing.

Now T is a concatenation (see section 1):

$$T = aa^*bb^*cc^*...zz^* \qquad (3)$$

where a, b, ... are 'keys' taken in any order from a known family $\mathcal{K}$ and a^*, b^*, ... are user supplied token lists; we call a^* the *argument* or *field* of the key a, and b^* the argument of b, etc...

The main step of parsing is to store a^*, b^*, ... in token list registers (or macros) associated to the keys a, b, ... When this parsing of T has been accomplished, TEX has a firm grip on

1) Recall that \special is one of the cohorts of \edef mentioned at the end of section 0.

the information encoded in T and typesetting can proceed.

Our purpose in this section is to propose ways to parse T in a few cases of practical importance.

First consider the specific example (1). We are just as happy (or happier) with the full expansion a^{**} of a^*, discussed at the end of section 0, since one can readily believe that in the specific context of (1) the expansion is unlikely to cause the sort of trouble mentioned there.[2)]

The argument a^* might well have TEX conditions and arithmetic (including =), while the full expansion a^{**} should be a dry number or dimension. In particular, it will not contain =, which we can then use as a tell-tale sign for a key.

We expand the whole of T (using `\edef`; see section 0) and note that this gives $aa^{**}bb^{**}\ldots$, i.e., the keys are intact. Since = is a tell-tale sign for the next key b, we can readily determine b. More precisely, in Rokicki's syntax, the keyword is delimited on the left by a space and on the right by =.[3)] We can thus split at b — or for greater speed use just the idea of formula $(*)$ in section 2 — to get a^{**} and $bb^{**}cc^{**}\ldots$. We store away a^{**} for key a, then iterate the process to get a grip on b^{**}, c^{**}, ... similarly.

In summary, in case the next key is always readily accessible, keyword parsing is a straightforward process. The time required seems then to be the least time for all the processes we will consider. Qualitatively speaking, the time per key is constant and independent of the number of keys.

The syntax discussed by Appelt [App, Chap. 5 (end)] is of this simple sort; his next keyword lies between the next semicolon and the next equal sign. (Appelt formulas nevertheless run through all keywords to find the next key, something to be avoided if there are many keys.)

The accessibility of the next key in Rokicki's case was probably a well-planned accident — related to Rokicki's driver wanting to parse this syntax in a hurry. In the absence of the tell-tale = above, or if we had wanted unexpanded arguments, the following general method would have worked.

Parsing process (A) — Substitution and self-analysis

This is a simple and general process that depends heavily on our token list processing in section 1. It is practical if there are just a few keys.

For each key k in $\mathcal{K}$, search[4)] for k in the toks T of form (3) and, if k is present, replace it (using splitting and concatenation of section 1) by the two tokens `\zzz\macro`k. Thus (after doctoring the extremities), we readily give altered T (in `\Ttoks`) the form:

$$\texttt{\textbackslash macro}a\ a^*\texttt{\textbackslash zzz\textbackslash macro}b\ b^*\texttt{\textbackslash zzz}\ldots \quad \texttt{\textbackslash macro}z\ z^*\texttt{\textbackslash zzz} \qquad (4)$$

This completes the substitution step.

Now for each k in $\mathcal{K}$, we are at liberty to define `\macro`k`#1\zzz` as a one-argument macro which places token list `#1` in a token list register (or macro) associated to k. Then writing `\the\Ttoks` as a command, we execute (4), and the result is to complete the parsing. The idea of this second step that we have subtitled 'self-analysis' has been used by Knuth in dealing with the TEX data structure called 'list' [Appendix D, p. 378–379].

Note that if there are N keys in $\mathcal{K}$, the parsing process always has N nontrivial steps and each applies to the whole token list. Thus the time of execution can be estimated as roughly proportional to nN where n is the number of keys actually present in the token list T. Consequently for N sufficiently large the process will be intolerably slow. How large? My tests suggest you should be worried for $N > 5$.

The $\mathcal{AMS}$-TEX bibliography reference macro `\ref` ... `\endref` is one that has well over a dozen keys; for it, one needs a better parsing technique. It has the peculiarity that the keys are all macros. Consider the example

```
\ref
 \key W \by A. Weil
 \paper Sur quelques
  r\'esultats de Siegel
 \journal Summa Brasil Math.
 \vol 1 \yr 1946 \pages 21--39
\endref
```

Note that only `\ref` has a balancing terminator `\endref`; it lets us scoop up the whole token list

2) The reason is that TEX always does this sort of expansion on the argument of `\special` before stuffing the result into the `.dvi` file for further processing by the printer driver. Clearly, the user will have himself to blame if he attempts for `\Special` what fails for `\special`!

3) To be more user-friendly, it would be advisable to allow space between (for example) `hsize` and =. Although this may double the time to locate the next keyword, one still does not have to run through all the keywords.

4) If one key is a subset of another, e.g., "SCALE" and "VSCALE", deal with the larger one first.

from \key to 21--39 as a macro argument. Once again, we have a parsing problem as described for (3).

There are six keys here: \key, \by, \journal, \vol, \yr, \pages. But parsing process (A) above would require searching with well over twice as many keys. The feature that each key is a control sequence lets us use a new process (B) which will be given in full detail.

Parsing process (B) — Sequestered self-analysis

This process usually applies when each key is a control sequence; it requires a few extra conditions which will become clear when the process has been described.

Since process (A) does indeed apply here, and what follows is comparatively difficult, I had better explain very clearly what (B) attempts to gain! Suppose that the set $\mathcal{K}$ of keys is big, say N of them (perhaps 25), and getting bigger year by year. We ask for a process that on a given example using n keys (perhaps $n = 5$) does not run substantially slower each year as N increases. We would like to get by with a few times n steps — to be more precise, not more than $a + bn$, where a and b are constants independent of N. In contrast, the similar estimate for (A) would be $a' + b'nN$. Thus in the usual succinct mathematical terminology, process (A) requires $O(nN)$ steps while (B) requires $O(n)$. The latter seems, qualitatively speaking, a 'nec plus ultra' of good behavior, because it means that the cost of parsing per field actually present is constant.[5)]

The 'box register' alternative to token list parsing that is actually used by $\mathcal{AMS}$-TEX for the \ref...\endref macro system enjoys the sort of linearity that we are promising for (B). On the other hand, the keyword parsing provided by W. Appelt [App, Chapter 5 (end)] simply does not apply.

The idea for (B) is to make a preliminary pass over the material between \ref and \endref to determine, for each key $\backslash k'$ that is present, the key $\backslash k$ that follows, and then define a macro $\backslash k'$#1$\backslash k$ (with argument #1 and delimiter $\backslash k$), which will, on a second pass, serve, much as in (A), to sweep up the field of $\backslash k'$ and store this toks in a corresponding macro expansion. For this first pass, subtitled 'sequestered self-analysis', one assigns special temporary definitions to each key to carry out this plan. A major difficulty is that I cannot prevent extraneous typesetting activity during the first pass; the best remedy known is to 'sequester' this extraneous material in an \hbox and annihilate it. Unfortunately, this \hbox involves a grouping that entraps definitions — unless one uses some global definitions. Perhaps surprisingly, this secondary difficulty is overcome without losing the expected behavior of the parsing process with respect to TEX grouping, namely that it change nothing outside braces enclosing the whole process.

Now we get down to programming process (B). The functioning prototype is given in a sidebar, but will probably have to be understood as it was built — by stages. The programmer has to define for each key $\backslash k$ in $\mathcal{K}$ (say \paper, to be specific) an artificial expansion that combines $\backslash k$ (say \paper) with key $\backslash k'$ (say \author) stored as author in a register called (for good reason!) \LastKeytoks@. The definition of \paper goes roughly as follows.

```
\def\paper
 {\global\def\authorAgent@
  {\def\author####1\paper
    {\def\authorBag@{####1}\paper}%
     \global\let\authorAgent@=\relax}%
   \LastKeytoks@={paper}%
   \aftergroup\authorAgent@
   \def\paper{\errmessage
    {  *** A key has been used
    twice. Once is max. ***}}%
 }                                    (5)
```

The programmer unfortunately is not in a position to write something so explicit — for example he does not know the actual name of the key that will precede \paper. Standard indirect methods involving \csname...\endcsname apply nevertheless. This macro depends on $\backslash k$ in a very simple way; so the TEXpert can get away with writing just one (nasty) macro \SetKeyDef@ (see sidebar) so designed that executing \SetKeyDef@{k} for $\backslash k$ in $\mathcal{K}$ sets things up once and for all.

To facilitate the parsing we use an extra terminal key \t@il, as well as an initial key \he@d.[6)]

One then executes:

```
\LastKeytoks@={he@d}
\setbox0=\hbox{\the\Ttoks\t@il}
\setbox0=\hbox{}
\let\t@il=\relax
\he@d\the\Ttoks\t@il                  (6)
```

5) In contrast, I do not know how to use TEX to reverse the order of a list of n tokens in $O(n)$ steps!

6) The (category 11) @ in he@d and t@il keep these out of the user's way. Likewise, \Ttoks, \e, \n, \cs, \ecs should be protected elsewhere.

The first appearance of \the\Ttoks is a dirty trick! What we really wanted to do is execute in order just the keys $abc\ldots$ found in T. But, as these are buried in T and not directly accessible, we execute all of T instead. This (unfortunately) causes spurious typesetting activity, but we catch the detritus in \box0 and annihilate it! [7)]

The global definitions are essential to pass information out of the first \hbox{...} group in (6), using \aftergroup. This is accomplished as follows. The tokens \aftergroup\authorAgent@ occur inside the grouping of \hbox{...} and cause the globally defined macro \authorAgent@ to be executed after the closing brace. This in turn prepares a second definition of \author for a second execution of \the\Ttoks in the last line.

This second execution will be similar to the last step in (A): the macro \author (new definition) will cause the field of the key \author to become the expansion toks of \authorBag@ — *outside* the \hbox{...} group. This process is carefully designed to cause no net global changes in case it occurs within a larger group; in particular \authorAgent@ is globally \relax before and after. The resulting 'escape from braces' without net global change seems in itself a worthwhile trick.

Both executions of \the\Ttoks cost $O(n)$ steps. While the first involves futile typesetting, the second is fast and purely syntactical. Let us have a closer look at the second: after a couple of expansion steps, what TEX sees is $aa^*bb^*cc^*\ldots$; then the first three tokens $a\,a^*b$ act together to put the field a^* in the expansion of the macro[8)] aBag@ leaving behind $bb^*cc^*\ldots$. Then bb^*c act together, and so on until all fields have been 'bagged' and only \t@il is left, which evaporates as we have set it equal to \relax.

The use of \aftergroup restricts the number n of keys used in any one parsing example to be less than the size of TEX's save stack space. This may mean $n < 100$ in current implementations of TEX; but soon your limitations should be much more liberal; already, OzTEX's configuration file of 1990 lets one push n up to 2000, and the total number N of keys to about 5000. Perhaps squeamishness about the use of \aftergroup [p. 374] can be relegated to the past. In any case, \aftergroup could be replaced by some global definitions without prejudicing the linear performance we have achieved.

This process (B) does have some drawbacks beyond the fake typesetting. (You expect a dirty trick to have some!)

(i) It assumes that T is fit to be put in an \hbox, and that, on execution of T, the key macros a, b, ... will be executed in that order.

This is a very mild restriction that refers to the first pass; it should in practice hold if each key field is fit to be composed on its own. If (i) is not satisfied, one can hope it will be if one suitably alters the TEX environment for the first pass.

(ii) The speed of parsing is a bit disappointing to me; I get about 50–100 key fields parsed per second with a 1987 microcomputer (16 mhz and 32 bit bus). In implementing (B), one has a great deal of latitude in programming style; perhaps I have made some bad choices; if so I hope some reader will offer better coding. This slowness may not be a serious fault if you have a sufficiently fast computer, or if this parsing is not going to be proportionally a major activity of TEX, or if the other TEX material is already slow to process — for example commutative diagrams, tables, or verbatim material.

A good feature of (B) that I did not expect is the brevity of the coding.

In summary, the one rather general parsing process (A) is firmly based on our token list processing, and is delightfully simple and safe, but, used with a large number of possible key options, it becomes slow. That has led us to consider process (B), whose time cost per key field actually parsed is essentially constant[9)] and independent of the total number of possible keys. In practice it seems that (B) is faster than (A) for $N > 5$.

```
% TestbedB.tex

%% Sidebar: Testbed for parsing method (B)
%% L. Siebenmann 1991

\chardef\CatAt\the\catcode`\@\catcode`\@=11
\newtoks\Ttoks@ \newtoks\LastKeytoks@
\let\e=\expandafter \let\n=\noexpand
```

7) One naturally wonders whether there is a much neater trick. One can marginally speed up this trick using the 'dummy' font device of the last dirty trick 9 in [Appendix D, p. 401], but if you are not careful you will instead lose time through overhead.

8) I have not used a token list register here to store the key field; this permits the number of keys to exceed the total number (256) of token list registers!

9) This assumes fields of constant size; if not, the dependence of time cost per field on the size of the field is more or less linear, with a substantial positive constant term.

```
\let\cs=\csname \let\ecs\endcsname
%% Texpert protect \e,\n,\cs,\ecs with @

\def\SetKeyDef@#1%
 {\e\def\cs#1\ecs{\MasterMacro@{#1}}}

\def\MasterMacro@#1%
 {\e\xdef\cs\the\LastKeytoks@ Agent@\ecs
   {\def\e\n\cs\the\LastKeytoks@\ecs
          ####1\e\n\cs#1\ecs
      {\def\e\n\cs\the\LastKeytoks@
                 Bag@\ecs
          {####1}\e\n\cs#1\ecs}%
    }%
    \e\aftergroup
        \cs\the\LastKeytoks@ Agent@\ecs
    \LastKeytoks@={#1}%
 }

\def\@K.#1.{\SetKeyDef@{#1}}

\@K.by.\@K.key.\@K.paper.\@K.jour.\@K.yr.
\@K.pages.\@K.issue.\@K.no.\@K.vol.
\@K.publ.\@K.eds.\@K.bysame.\@K.paperinfo.
\@K.book.\@K.publaddr.\@K.lang.
\@K.bookinfo.\@K.finalinfo.\@K.t@il.

\def\ref#1\endref{\Ttoks@={#1}%
 \LastKeytoks@={he@d}%
 \setbox0=\hbox{\the\Ttoks@\t@il}%
 \let\t@il\relax
 \e\he@d\the\Ttoks@\t@il\Typeset@}

\let\Typeset@\relax % stop after parsing

%% begin test
\def\R{% for time test
 \ref
  \key W \by A. Weil\paper Sur quelques
                    r\'esultats de Siegel
  \jour Summa Brasiliensis Math.
  \yr 1946 \pages 21-39
 \endref}

\def\RR{\R\R\R\R\R\R\R\R\R\R}
\def\RRR{\RR\RR\RR\RR\RR\RR\RR\RR\RR\RR}

\show\key % all set?
\RRR % do 100 iterations

%\show\keyBag@    \show\byBag@ % checks?
%\show\paperBag@ \show\jourBag@
%\show\yrBag@     \show\pagesBag@
\catcode`\@=\CatAt % restores catcode
\end
```

Should token list parsing have fixed the bug in the $\mathcal{A}_{\mathcal{M}}\mathcal{S}$-TEX reference macros?

As we will see presently, the answer is no, but it seems worth examining the pros and cons since many of them would have to be examined in any large scale application of parsing based on token lists.

The bug, located in `amsppt.sty` (versions $\leq$ 2.0), prevented hyphenation after explicit hyphens, or after mathbins and mathrels, for line-breaking in references. As Michael Downes so nicely explained [Do], this bug (and also the residual problems with Knuth's fix) have occurred because, if a fragment of a reference is put into an hbox or even a vbox, certain stages of line-breaking may be done prematurely and hence inappropriately in that box.

The plan for using token registers is very simple. Place the various parts of a single bibliography reference into as many token lists using parsing method (B), edit these token lists as necessary using the text processing of section 1, concatenate them in the desired order[1] to make a single token list for the reference in question, ultimately releasing the whole reference 'en block' into TEX's intestines for typesetting.

The bug will not occur since one replaces the troublesome boxes by token registers. Indeed those aspects of typesetting related to line breaking simply do not take place in token registers; they are delayed until the full reference is ready for typesetting.

Ron Whitney and Mike Downes tell me that the idea of using token lists in place of boxes was well known but considered to be an impasse (no way!).

The token list approach has some intrinsic advantages over the box-oriented approach. We have already mentioned the possibility of doing some text editing before printing (say to replace AMS by Amer. Math. Soc.). In this vein, there is the possibility, not well afforded by the box approach, of having any key's data influence the action taken for any other, for example, when the reference is

1) Such ordering should have a cost proportional to $n \log n$ when n out of N keys are present. But in $\mathcal{A}_{\mathcal{M}}\mathcal{S}$-TEX a cost aN with a very small is tolerated instead for simplicity.

a book the style of several entries could reasonably change. One can also output the references to a file in a convenient 'data structure' format, to facilitate further processing. This might, for example, facilitate preparation of a citation index (or other bibliographic data base) for journals using $\mathcal{A}\mathcal{M}\mathcal{S}$-TEX. As this idea applies to reprocessing archived articles, the toks parsing approach may ultimately be complementary to the box-oriented approach even in the specific context we are considering. Another advantage already mentioned is the virtual inexhaustability of token registers: although there exactly 250 token registers or box registers in the strict sense, macros, of which there are thousands available, can be employed as auxiliary token registers; they indeed were in the testbed for (B).

Nevertheless, we will have to wait for some future occasion to see a large-scale test of the above token-parsing ideas. There are a host of reasons that, taken together, are quite cogent. Repair of `amsppt.sty` (where the faulty macros reside) has already been successfully made by Mike Downes (for `amsppt.sty` version 2.1 of July 1991) using Knuth's `\vbox` approach plus extra work to suppress undesirable side-effects. A very practical consideration is that Knuth's approach is comparatively close to Spivak's, so that much less rewriting of this hefty complex of macros was required.

Furthermore, the box approach is exceedingly fast — to the point that bibliographies are composed faster than most mathematics. This turns out to be more than twice as fast as the next fastest contender, the parsing approach (B), which is in turn more than twice as fast as (A). See [p. 385].

Finally, there is a general weakness (mentioned on [p. 381–382, p. 385]) afflicting all macros having arguments which are blindly scooped up as chunks of input, namely: *category changes within the arguments will be ignored because category is fixed at input.* We did propose to scoop up T above using `\ref#1\endref`! One impact is that, with our present approach, `\verb` (of LATEX) and `\lit` (of $\mathcal{A}\mathcal{M}\mathcal{S}$-TEX) would become inoperative within a reference.[2)] This may prove annoying, but one can live with it by 'hand setting' or by importing literal material in a box register. Another impact is that language changes would have to be made so as not to involve category changes, which fortunately is possible. It might be be desirable to set up some warning using `\message{...}` to be triggered by uses of `\catcode` within `\ref...\endref` and give indication of alternatives. These category problems are annoying but they are not debilitating.

In summary, token list parsing in version (B) compared to the box register alternative seems a promising alternative because of multiple hitherto unused possibilities we have mentioned; it equals box registers in dealing 'linearly' with increasing loads, but is always slower by a small integer factor; and finally it may, alas, be penalized for blocking category change. I rate that an honorable second place on a tough course.

The role of token registers in Plain TEX, LATEX, or $\mathcal{A}\mathcal{M}\mathcal{S}$-TEX was quite marginal. But recently, their role has become quite significant, for example in M. Spivak's $\mathcal{A}\mathcal{M}\mathcal{S}$-TEX (released recently into the public domain). Now that TEX is no longer evolving, I expect TEX programming will still advance a long way by increasingly calling upon currently underused resources.

2) One can *in principle* pick up the token list T one token at a time watching for `\verb` and `\lit`, in order to avoid this weakness, but that sort of procedure is much too slow. For just this reason we have ignored parsing procedures based on sequential token-by-token examination, using `\futurelet`.

Bibliography

[**App**] W. Appelt, *TEX für Fortgeschrittene*, Programmiertechniken und Makropakete, Addison-Wesley, Bonn, 1988.

[**Do**] M. Downes, Linebreaking in `\unhboxed` text, *TUGboat* 11, no. 4 (Nov. 1990) 605–612.

[**Kn**] D. Knuth, *The TEXbook*, copyright 1984, Amer. Math. Soc., Volume 1 of *Computers and Typesetting*, Addison-Wesley.

[**Wh**] R. Whitney, Sanitizing control sequences under `\write`, *TUGboat* 11, no. 4 (Nov. 1990) 620–622.

◇ L. Siebenmann
Matématique, Bât 425
Univ. de Paris-Sud
91405 Orsay, France
`lcs@matups.matups.fr`

Macros

The bag of tricks

Victor Eijkhout

Hello everyone.

Here is the second installment of the bag of tricks, ready-to-use macros without the bother of needing to understand them. (And contributions from the *TUGboat* readership for this column are still welcome.)

When you start fiddling around with boxes, pretty soon you run into the fact that a `\vbox` is almost immediately as wide as the whole page, even if there is only a single character in it: `\vbox{a}`. Attempts such as `\vbox{a\par}` are not much of an improvement. But supposes you want a box to be as wide 'as it really is', for instance

```
a\par b          a
c\par            b c
d        gives   d     as output,
```

then something else is needed. The input for this example was (I have set `\parindent` first to zero):

```
\snugbox{\begin{verbatim}a\par b
c\par
d\end{verbatim}}\quad
gives\quad
\snugbox{a\par b c \par d}\quad
as output,
```

Here is the source for the `\snugbox` macro, to be placed
between `\catcode'@=11` (or `\makeatletter`) and `\catcode'@=12` (or `\makeatother`):

```
\def\snugbox{\hbox\bgroup
    \setbox\z@\vbox\bgroup
        \leftskip\z@
        \bgroup\aftergroup\make@snug
        \let\next=}
\def\make@snug{\par\sn@gify\egroup
    \box\z@\egroup}
\def\sn@gify
   {\skip\z@=\lastskip \unskip
    \advance\skip\z@\lastskip \unskip
    \unpenalty
    \setbox\z@\lastbox
    \ifvoid\z@ \nointerlineskip
         \else {\sn@gify} \fi
    \hbox{\unhbox\z@}\nointerlineskip
    \vskip\skip\z@
    }
```

This macro is not completely fool-proof, but it works in a lot of cases, for instance it contains a `\snugbox` to contain several paragraphs.

The example used above did not look very pretty. Wouldn't it be better if the input and output were centered vertically with respect to 'gives' and 'in the output'? Using `\vcenter` this would look like

```
$\vcenter{\snugbox{ ... }}$\quad
gives $\vcenter ...
```

But typing all these dollars is a bit tiresome. Also you may know that you cannot write

```
\setbox\mybox\vcenter{...}
```

Both of these points are remedied by the following macro:

```
\def\textvcenter{\hbox\bgroup$
    \everyvbox{\everyvbox{}%
          \aftergroup$%
          \aftergroup\egroup}
    \vcenter}
```

And now you write

```
\textvcenter{\snugbox{
\begin{verbatim}a\par b
c\par
d\end{verbatim}}}\quad gives ...
```

for

```
a\par b
c\par     gives ...
d
```

Simple, isn't it?

See you next time, and keep those braces balanced!

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

Oral TEX: Erratum

TUGboat **12, no. 2, p. 272–276**

Victor Eijkhout

Alert reader Bernd Raichle pointed out that my macro for lexicographic ordering was not correct. Here is a repaired version. Replace the definition of `\ifallchars` at the bottom of page 274, column 1, by the following.

```
\def\ifallchars#1#2\are#3#4\before
   {\if#1$\say{true\xp}\else
     \if#3$\say{false\xp\xp\xp}\else
      \ifnum`#1>`#3 \say{false%
         \xp\xp\xp\xp\xp\xp\xp}\else
       \ifnum`#1<`#3 \say{true%
         \xp\xp\xp\xp\xp\xp\xp
         \xp\xp\xp\xp\xp\xp\xp\xp}\else
       \ifrest#2\before#4\fi\fi\fi\fi}
```

This macro contains the slightly ridiculous sequence of 15 `\expandafter` commands. However, Bernd Raichle also supplied his own solution to string testing and lexicographic ordering, which use a somewhat different principle (and are in addition shorter) than mine.

```
\def\ifsamestring#1#2{\csname
     if\allchars#1$\are#2$\same
     \endcsname}
\def\allchars#1#2\are#3#4\same{%
  \if#1$%
    \if#3$true\else false\fi
  \else
    \if#1#3\allchars#2\are#4\same
    \else false\fi
  \fi
}
\def\ifbefore#1#2{\csname
     if\allchars#1$\are#2$\before
     \endcsname}
\def\allchars#1#2\are#3#4\before{%
  \if#1$%
    true%
  \else \if#3$%
    false%
  \else \ifnum`#1>`#3
    false%
  \else \ifnum`#1<`#3
    true%
  \else
    \allchars#2\are#4\before
  \fi\fi\fi\fi
}
```

Some Basic Control Macros for TEX

Jonathan Fine

Abstract

This article is concerned with the mouth of TEX, particularly macros and the primitives `\if...`, `\else` and `\fi` used to control expansion. (Recall that the mouth expands the input stream until it comes to something unexpandable, which is then passed to the stomach.)

Although it can do little but absorb parameters and expand macros, the mouth is powerful. Alan Jeffrey (*Lists in TEX's Mouth*, *TUGboat* 11, no. 2 pp. 237–245, June 1990) shows that it can do the lambda calculus.

Our purpose is more limited. It is to define and describe macros `\break`, `\continue`, `\switch`, `\return`, `\exit`, `\chain`, and labels `\end` and `':'` that make it easier to write TEX macros. These macros will be collected into a file `control.sty`.

Several worked examples are given.

1 Introduction

There are now some substantial programs written in TEX. The source for LATEX runs to 8,500 lines. PICTEX has 3,500 lines. A style file might have 120 lines of code and 300 lines of comments. TEX is a terse and at times cryptic language. A great deal can be done in 26 short lines. This article is devoted to making life easier for that suffering creature, the writer of TEX macros.

Acknowledgements. The author thanks the referees for their careful comments, which have greatly improved the article.

1.1 Ignoring spaces

TEX has rules for ignoring spaces in the input stream that are well adapted to reading a text file spiced with control sequences. But these rules do not suit the macro writer, whose words are few, and control sequences many. Many programming languages today are 'free form'. White space is ignored, allowing the programmer to indent or otherwise arrange the code, so that the meaning is more easily read.

Accordingly, by changing the category of tab, carriage return, space, and `'~'`

```
\chardef\ignore 9
\catcode 9\ignore
\catcode13\ignore
\catcode32\ignore
\catcode`\~ 10\relax  % make ~ space
```

we ignore all white space except when we explictly ask for it.

1.2 Speaking clearly

All m@nn@r @f d@vic@s @re us@d t@ gener@te priv@te c@ntr@l s@qu@nc@s in m@cro fil@s. I'll say that again. All manner of devices are used to generate private control sequences in macro files. And they are a nuisance. Here, we will take the programmer's side, and use names built out of ordinary letters.

We also give our control sequences the names we *want* to give them. *In particular* `\break` *and* `\end` *do not have the usual meanings.*

1.3 Top-down and step-wise

The author hopes that he has used here the techniques of *top-down programming* and *step-wise refinement* to obtain these basic control macros.

This means we first identify and solve some of the essential features of the problem, and then go to details. We may have to go round several times until we are finished.

1.4 Auxiliary macros

We will need some general purpose helper macros.

```
\def\unbrace #1 { #1 }
\def\gobble #1 { }
\def\gobbletwo  #1 #2 { }
```

2 Macros for loops

Repetition and termination are the essential features of a loop.

2.1 Writing `\myloop`

Our first attempt

```
\def\myloop {
  \myloop
}
```

produces a loop that will, when executed, endlessly do nothing useful. It has the required virtue of repetition, but in excess.

First, we will make the loop do something useful. (Then worry about stopping.) Suppose, for example, that we wish `\myloop abc...` to have

```
\myoperation a
\myoperation b
\myoperation c
  ...
```

as its result.

We add a parameter and an action to the definition of `\myloop`.

```
\def\myloop #1 {
  \myoperation #1
  \myloop
}
```

Now worry about stopping. Suppose that `\myloop` is to stop when some token, for example `\end`, is passed as a parameter. Before executing `\myoperation #1` we must make a test `\ifx#1\end` and if it succeeds, the loop is to be terminated.

Termination will require a strange trick which we will call `\break`. (The programming language *C* uses `break` for a similar purpose.)

We now have

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

where `\break`, when called, terminates the loop.

2.2 `\breaking` from `\myloop`

We are now able to specify and code the `\break` command. Its definition is at first less than intuitive, although arrived at logically. To prevent repetition, `\break` must absorb the lines

```
\myoperation #1
\myloop
```

of `\myloop`. We can do this by letting `\myloop` delimit the argument to `\break`. The command

```
\def\break #1 \myloop { }
```

will absorb the unwanted tokens. Sadly, it also absorbs the `\fi`, which we must put back again. This is easy! Just put it back.

```
\def\break #1 \myloop { \fi }
```

To summarize, the code for `\myloop` works because

- The expansion of `\myloop` terminates with `\myloop`. (This is tail recursion.)
- The expansion of `\break` successfully breaks the loop.

2.3 Don't do this

Someone coding their first loop might write

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

which looks correct but isn't. The reason is subtle. We will expand `\myloop AB`, assuming that `\myoperation` is `\gobble`. Here it is, step-by-step.

1. `\myloop AB`
2. `\ifx A\end \else`
   `\myoperation A\myloop \fi B`
3. `\myoperation A\myloop \fi B`
4. `\myloop \fi B`

and now we are in trouble. `\myloop` is about to eat the `\fi`. It should be getting the B. In fact `\myloop` will continue to generate and consume `\fi` tokens, and will never get to B.

5. `\ifx \fi \end \else`
   `\myoperation \fi \myloop \fi B`
6. `\myoperation \fi \myloop \fi B`
7. `\myloop \fi B`

For `\myloop` *to be successful, calling itself must, literally, be the last thing it does. Only then can it read the next token, which is* B *in our example.* Computer scientists call this 'tail-recursion'. This trick avoids another hazard, the filling up of memory during a long loop. **5.1** gives an example of how this can arise (see also *The TEXbook*, p. 219).

2.4 Coding Tail Recursion

It is traditional to use an assignment to a scratch control sequence

```
\def\myloop #1 {
  \ifx #1\end
    \let\next\relax
  \else
    \myoperation #1
    \let\next\mymacro
  \fi
  \next
}
```

to achieve tail recursion.

Assignments (and other unexpandable primitives) are not performed within `\edef`, `\xdef`, `\message`, `\errmessage`, `\write`, `\mark`, `\special` and also the `\csname`–`\endcsname` pair. This limits the usefulness of the traditional design.

However, the macros of `control.sty` can safely be used in these situations, and also when TEX is looking for a number, dimension, glue or filename.

2.5 Writing `\yourloop`

Now suppose you wish to use the above to code another loop, `\yourloop`. A problem appears. The definition

```
\def\break #1 \myloop{ \fi }
```

has `\myloop` coded into it, and so is not suitable for coding `\yourloop`. We do not wish each loop to need a different `\break` command. This would be wasteful. We notice that the key to `\break` is that it gobbles to a certain point, and then puts down a balancing `\fi`. Here is a first guess to a universal `\break`.

```
\def\break #1 : { \fi }
```

(*C* uses the colon ':' as a label to allow use of the much-abused `goto` command.)

Given this `\break`, the definition

```
\def\yourloop #1 {
  \ifx #1 \end \break \fi
  \youroperation #1
  :\yourloop        % notice the colon!
}
```

is natural. However, as we have introduced a ':' into each iteration of the loop, we should ensure that its expansion is empty. (`plain` has `\def\empty{}`.)

```
\catcode`\: \active
\let : \empty
```

The rules around the code indicate that it is to be part of the macro file `control.sty` and not an example.

There is another failing—`\break` will gobble

```
\youroperation #1
:
```

and leave

```
\yourloop        % notice the colon!
```

which is the `continue` command in *C*! To be successful, `\break` must consume also the token that follows the ':' delimiter.

(The author expects `\continue` will be used less often than `\break`. It causes the next iteration of the loop to begin. For example, to process only some of the input tokens, code similar to

```
\def\ignoresome #1 {
  \ifignore #1 \continue \fi
  % now process those tokens that
  % have not been ignored
  :\ignoresome
}
```

should be used, where `\ifignore` determines the fate of the token.)

The definitions

```
\def\break #1 : #2 { \fi }
\def\continue #1 : { \fi }
```

will be refined no more in this article.

3 Use \switch or \else!

Here we construct in TEX an analogue to the switch construction provided by *C*. It is useful when one of a list of cases is selected, depending on the value of some quantity. (Note that \switch does not share with *C* the *fall through* property. It is more like the CASE construction in Pascal.)

3.1 The alphabetic \fruit macro

Suppose we wish to write a macro \fruit such that \fruit a will result in \apple, \fruit b in \banana etc. One method is to produce a cascade of \if...\else...\fi statements. However, we could write

```
\def\fruit #1 {
  \switch \if #1 \is
      a \apple
      b \banana
      c \cherry
      d \date
  \end
}
```

if only we had a suitable \switch command. We will produce such a command. (The reader may benefit from trying to write such a command before reading on.)

First, some terms.

'\if #1' is the *test*
'a \apple' is the first *alternative*
'a' is the *key* to the first alternative
'\apple' is the *option* for the first alternative

It is clear that \switch must go through the alternatives one after another, reproducing the *test*

```
\def\switch #1 \is  %  the test
       #2 #3        %  key & option
{
  ...
  \switch #1 \is    % reproduce
}
```

and doing nothing unless the *key* fits the test

```
\def\switch #1 \is  %  the test
       #2 #3        %  key & option
{
  #1 #2 ... \fi  %  test key
  \switch #1 \is
}
```

in which case we should

- gobble to the end (marked by \end) of the expansion of \fruit
- insert the current *option* #3

and so we have

```
\def\switch #1 \is
       #2 #3
{
  % if  ( test key )  succeeds
  #1 #2 \exit #3 \fi  % do option
  \switch #1 \is
}
```

where \exit is a helper macro for \switch.

3.2 An \exit for \switch

As with \continue, \exit must gobble to some point and restore the \fi balance

```
\def\exit #1 \end { \fi }
```

but it should also pick up and reinsert the current option

```
\def\exit #1 #2 \end { \fi #1 }
```

which will work in the context of \fruit. It will fail if the option has several tokens.

For example, the definitions above expand

```
\switch \if a \is a {Jonathan} \end
```

to 'J'.

There are several solutions to this problem. Here is the one that executes the most rapidly.

```
\catcode`\@ 11^  %  make  @  a letter
                 %  Section 1.2  lies
\let \@fi \fi
\def\switch #1 \is #2 #3 {
  #1 #2 \@exit #3 \@fi
  \switch #1 \is
}
\def\@exit #1 \@fi #2 \end { \fi #1 }
\catcode`\@ 12^   %  put  @  back again
```

where \@exit and \@fi are helper macros, private to \switch.

3.3 Default actions for \switch

The expansion of '\fruit z' will fail horribly. As 'z' is not a key, \switch will read and discard up to the 'd \date' alternative, and then read \end and another parameter from the input stream. Now we are in trouble. \switch is still expanding, and there is no \end in sight.

Unless a matching key is sure to be found, a \switch should have a line handling the default. If \nofruit is to handle the default for \fruit, the line

```
#1 \nofruit
```

should be inserted as the last alternative.

(Another method would be to have \switch test for the \end token before reading '#1' and '#2'. Using macros to do this would result in a much slower \switch. But see section **11.**)

4 Applying \switch to \markvowels

By way of an example, we apply \switch to a problem posed and solved in Norbert Schwarz's *Introduction to TEX*, Ch7 §7.

4.1 The problem

We wish to write a macro \markvowels that prints the vowels of a given word in a different typeface. For example

```
\markvowels audacious.\endlist
```

is to give

audacious.

(There is a subtle reason why we use \endlist rather than \end. There is a surprise in the expansion of a \switch that has \end as a key.)

4.2 The solution

Here are some pointers for the solution.

- We need a \switch whose keys are a, e, i, o, u, \endlist and the default handler #1.
- Every letter, vowel or not, is to be printed.
- If a letter is a vowel, we apply \enbold to it.
- The expansion of \markvowels is to finish with \markvowels.
- When an option is selected, all up to the \end of the \switch is gobbled. For the key \endlist the two tokens #1 \markvowels must be absorbed.
- We are not obliged to use ':' when constructing a loop.

And here it is

```
\def\markvowels #1 {
  \switch \ifx #1 \is
      a \enbold
      e \enbold
      i \enbold
      o \enbold
      u \enbold
      \endlist \gobbletwo
     #1 \empty
  \end
  #1 \markvowels
}
```

with helpers

```
\def\enbold #1 {{ \bf #1 }}
\def\endlist { \gobble \endlist }
```

5 Finite State Automata (FSA)

The stomach of TEX, as the reader must well be aware, can be in one of number of states—horizontal mode, vertical mode, etc. The result of a command, such as \hbox{A}, will often depend on the current state. There are also rules that govern the transition from one state to another. Similarly, the text of a document passes from state to state—ordinary text, quotation, theorem, list item, and so forth. LATEX does this by changing the environment.

One way of coding such a device is to let the state be represented by a macro or parameter, whose value is then tested or altered by a single macro that contains code for *all* of the automaton's states. Although such a design is not without merit, here we will code Finite State Automata by using one macro for each state.

5.1 Skipping multiple blank lines

We proceed by means of an example. Suppose that we are \reading a file, and that we wish to ignore all but the first of adjacent blank lines. We have two states.

- \lastlineblank
- \lastlinenotblank

Here is a first attempt to code the states.

```
\def\lastlineblank {
  \read\thefile to \currentline
  \ifx\currentline\blankline
    % do nothing, call same state
    \lastlineblank
  \else
    \process\currentline
    \lastlinenotblank
  \fi
}

\def\lastlinenotblank {
  \read\thefile to \currentline
  \ifx\currentline\blankline
    \processblankline
    \lastlineblank
  \else
    \process\currentline
    \lastlinenotblank
  \fi
}
```

Although the above works for small files, it has a fault. Each time a line is read, the number of unbalanced \fis increases by one. The missing \fis (and other code) are pushed into the input stream, and will produce

```
! TeX capacity exceeded,
    sorry [input stack size=200].
```

before too long.

5.2 Using \end to \return a state

This problem arises because the next state is called before the current state is finished. As in \switch, at the end of each state macro we will place an \end marker, and use \return to move the next state to the head of the input stream. (*C* uses return to terminate a function with a specified value.)

Here we go. \lastlineblank should be

```
\def\lastlineblank {
  \read\thefile to \currentline
  \ifx\currentline\blankline
    \return\lastlineblank
  \else
    \process\currentline
    \return\lastlinenotblank
  \fi
  \end
}
```

where \return

```
\def\return #1 #2 \end { \fi #1 }
```

will gobble to the \end of the current state, balance the \fi, and place the next state at the front of the input stream.

To end the current state and do nothing more, the command \exit

```
\def\exit #1 \end { \fi }
```

should be called.

The command \end is merely a delimiter. We define

```
\let \end \empty
```

so that no harm occurs should it be executed.

5.3 Dealing with end-of-file

The code above continues to \read, even when the file has come to an end. An elegant solution is to write

```
\def\lastlineblank {
  \readfile\thefile\currentline\exit
  \ifx\currentline\blankline
    \return\lastlineblank
  \else
    \process\currentline
    \return\lastlinenotblank
  \fi
  \end
}
```

where \readfile takes three parameters.

- #1 an input stream number.
- #2 *the macro the stream is to be read to.*
- #3 the action to be taken on end of file.

Here is \readfile (see also **7** and **9.5**).

```
\def\readfile #1 #2 #3 {
  \ifeof #1
    % #3 may be several tokens
    % to be safe, we brace it
    \return { #3 }
  \else
    \read #1 to #2
  \fi
  \end
}
```

6 Choosing between ':' and \end

The delimiters ':' and \end perform similar but different functions. The programmer is advised on their use, and introduced to the last control macro, \chain.

6.1 The differences

':' and \end have the same \empty meaning. The difference is that ':' delimits \break and \continue, while \end delimits \return and \exit. Each of these macros will jump to ':' or \end, and put down a balancing \fi.

Although \break and \exit are analogues, \return has a flexibility that \continue lacks. We can (and must) decide what to \return but \continue provides no such choice.

To complete the use of ':' we introduce \chain. (See **9.5** for an example of its use.)

```
\def\chain #1 #2 : #3 { \fi #1 }
% Here ends control.sty .
% If you wish, restore white space.
```

6.2 Making the choice

Suppose that in the normal course of events, `\mymacro` will be followed by `\usualmacro`, where `\usualmacro` may or may not be `\mymacro`. Then the form

```
\def\mymacro ... {
    % code goes here
    % use \break, \continue
    % and \chain
  :\usualmacro
}
```

is preferred.

If there is no single most likely outcome, then

```
\def\mymacro ... {
    % code goes here
    % use \exit and \return
  \end
}
```

is probably best.

7 The `\fi` count problem

There is an error in `\readfile` that the author and the referees did not notice. However, when this macro was used, TEX found it.

We get the '`! Extra \fi.`' error. To understand why, suppose `\thefile` is at an end. `\lastlineblank` calls `\readfile` which `\returns` `\exit`. At this point the `\ifeof` in `\readfile` has been exactly matched by the `\fi` put down by `\return`. Now `\exit` gobbles to the `\end` and puts down *another* `\fi`. This is the error.

The problem is with the `\fi` count. `\exit` and the like put down `\fi`s to balance the ones they gobble. They have to do this, because TEX keeps in its main memory a record of each unbalanced `\if`. When the job is finished, they are reported. If TEX could be told not to do this, the balancing `\fi`s could be omitted and the problem would go away.

(The apology

```
! TeX capacity exceeded,
    sorry [main memory size=65533].
```

is produced when the macro

```
\def\doit { \iftrue \doit }
```

is executed.)

Given TEX as it is, it seems best to produce `\fi`-less verions of `\exit` and the like for precisely this situation. Replacing the `\exit` in `\lastlineblank` by

```
\def\gotoend #1 \end { }
```

will make the problem go away.

Or rather, this will move the problem. The macro writer now has to determine whether a balancing `\fi` is needed.

8 The Official version of `control.sty`

Here we list the version of `control.sty` that is to be used by macro writers. Compatibility with other macros demands that some changes be made.

- To allow active ':' to be used by other macro packages, ':' is made a letter, and throughout ':' is replaced by '`\::`'.
- Because the names `\break` and `\end` are already taken, uppercase names are used throughout.
- The `\fi`-less version of `\break` is to be `\BREAK`, while `\:BREAK` puts down a balancing `\fi`.

There is another problem — `\exit` and `\return` clash with `\switch`. All three macros use `\end` as a delimiter. This is not a desirable feature, and so `\SEND` (Switch-END) will be used to delimit `\SWITCH`.

Finally, by letting `\IS` equal `\fi` allows lines such as

```
\SWITCH \ifx #1 \IS
```

to be correctly skipped in conditional text (see *The TEXbook*, p. 211).

This article will now use these definitions.

```
\immediate\write16{ control.sty v1.0
 --- Jonathan Fine, 24 March 1991. }
\immediate\write16{ Public Domain, see
TUGboat (to appear) for documentation}

\catcode`\: 11     \catcode`\@ 11
\let\::\empty      \let\END\::

\def\BREAK#1\::#2{}
\def\CONTINUE#1\::{}
\def\CHAIN#1#2\::#3{#1}
\def\RETURN#1#2\END{#1}
\def\EXIT#1\END{}

\def\:BREAK#1\::#2{\fi}
\def\:CONTINUE#1\::{\fi}
\def\:CHAIN#1#2\::#3{\fi#1}
\def\:RETURN#1#2\END{\fi#1}
\def\:EXIT#1\END{\fi}

\let\@fi\fi        \let\IS\fi
\def\SWITCH#1\IS#2#3%
  {#1#2\@EXIT#3\@fi\SWITCH#1\IS}
\def\@EXIT#1\@fi#2\SEND{\fi#1}

```

26. `\catcode`\: 12 \catcode`\@ 12`

The author would like to receive examples of the use of these macros, and reports of problems and bugs.

As a general rule, before using a control macro that gobbles from *a control macro* **A**, to *a label,* `\END` *or* `\::` *as appropriate,* **B** start at **A** and read on until one reaches one of

- a `\fi`
- an `\if...`
- **B**

(but skip code enclosed by braces). In the first case use the `\fi`-ed version, otherwise the `\fi`-less.

9 Odds and Ends

Here are various bits and pieces that don't belong anywhere else. Some are quite important.

9.1 Name and Context

Other programming languages avoid conflict of names by giving each identifier a scope which is usually less than global. This is done by mapping each scoped identifier to a unique symbol, such as a number. I have work in progress that will add this capability to TEX. It will be a TEX macro package.

9.2 Nested conditionals

Because `\:BREAK` et al. replace only one gobbled `\fi`,

```
\if...
  \if...
    \:BREAK
  \fi
\fi
\::\nextmacro
```

will unbalance the `\fi`-count.

Rather than introduce `\::BREAK` it is better for the moment to say that such code is bad style, and discourage it. (The author would like to see any problem whose best solution requires breaking from a nested `\fi`.)

9.3 `\RETURN` or `\EXIT`

If the completed execution of `\mymacro` requires no parameters, *and* buildup of the input stack is not a problem, then instead of

```
\RETURN \mymacro
```

one can use

```
\mymacro \EXIT
```

which is slightly quicker. (`\EXIT` and any tokens between it and the matching `\END` will be sitting in the input stack waiting to be skipped until `\mymacro` has done its work.)

9.4 Dedicated `\SWITCH`

If large use is made of, for example,

```
\SWITCH \ifx #1 \IS
```

then it is better to use a specially adapted switch.

```
\def\SWITCHx #1 #2 #3 {
  \ifx #1 #2 \@EXIT #3 \@fi
  \SWITCHx #1
}
```

9.5 A better `\readfile`

In the expansion of `\readfile`, `#3` is read, copied into place, and then either thrown away or read and copied again.

In the normal course of events, `\readfile` needs only `#1` and `#2`. The end of file action `#3` will be discarded. Thus,

```
\def\readfile #1 #2 {

   \read #1 to #2
\::\gobble   % gobble '#3'
}
```

is a step towards the more efficient (and smaller)

```
\def\readfile #1 #2 {
  \ifeof #1
    \:CHAIN \unbrace
  \fi
  \read #1 to #2
\::\gobble   % gobble '#3'
}
```

Note that `\::` is very helpful, even though `\readfile` is not a loop.

10 Performance

It seems that once the idiom is mastered, these basic control macros will make it easier to write TEX macros.

The result will be code that is concise and relatively easy to understand. Code that is compact will load more rapidly from mass storage and use fewer words of memory.

It also seems likely that the idiom here will encourage utility commands, such as `\readfile`. This will reduce the size of both the code and the hash table.

Where speed of execution is paramount, custom devices are required. A carefully crafted cascade of `\if ... \else ... \fi` statements will run somewhat quicker than the `\switch` alternative. In other areas of programming, the prevailing wisdom is that good algorithms make for rapid execution.

Once the program is tested and running properly, significantly quicker performance can be obtained by rewriting a small amount of the code in lower level commands.

11 Enhancements to TEX

The Grand Wizard has said "no further changes except to correct extremely serious bugs" (*TUGboat* 11, no. 4, p. 489, June 1990) but this does not stop the wanting. That unbalanced `\if`s accumulate in memory without limit has already been mentioned.

Here are two devices that would improve the basic control macros of this article.

`\nil`—a primitive that does nothing. Although this is available as a macro, `\def\nil{}`, as an interpreted command it is over three times slower than the primitive `\relax`, which does slightly more!

`\abort`—a command which when passed as a parameter to a macro immediately halts its expansion. (If the macro is not `\long` then the *token* `\par` has the desired effect, but the error condition so generated is an unwanted side-effect.)

For the rest of the section, suppose that `\abort` has *both* of these properties. There are nice results. Provided we

```
\let \SEND \abort
```

the `\fruit` with no action as default becomes

```
\def\fruit #1 {
  \SWITCH \if #1 \IS
      a \apple
      ...
      d \date
  \SEND
}
```

which is more intuitive.

Provided `\END` is also set to `\abort`, it can be used to delimit `\markvowels`, which becomes fewer tokens executing faster.

```
\def\markvowels #1 {
  \SWITCH \ifx #1 \IS
      a \enbold
      ...
      u \enbold
      \SEND
      #1 \markvowels
  }
```

It is also simpler. The somewhat obscure line

```
\endlist \gobbletwo
```

is no longer needed.

Finally, we (almost) have an elegant means of handling default values.

```
\let \DEFAULT \abort
```

allows

```
\def\fruit #1 {
  \SWITCH \if #1 \IS
      a \apple
      ...
      d \date
      \DEFAULT \nofruit
  \SEND
}
```

to code a default value of `\nofruit`.

(There are two problems here. If `\nofruit` expects a parameter, it will get `\SEND`, which will then `\abort` it! This is wrong. In this situation

```
\def\USE #1 \SEND { #1 }
```

will allow

```
\DEFAULT \USE \nofruit
```

to correctly code such a default. The second problem is more serious. Should a key satisfy the test, such a `\DEFAULT` will `\abort` the `\@EXIT` macro called by `\SWITCH`. This is wrong.)

This area needs further investigation.

12 A Groaning Pun

Asked to write a macro `\goodthing` that does something useful, the design

```
\def\goodthing ... {
  ...
  \END
}
```

was used by 7 out of 10 programmers.

This only goes to show that most `\goodthings` come to an `\END`.

◇ Jonathan Fine
203 Coldhams Lane
Cambridge CB1 3HY
England
Tel +44 223 215389

Self-replicating macros

Victor Eijkhout and Ron Sommeling

The problem of writing a program that gives its source as its output is one of the oldest conundrums of computer science. (An extended discussion can for instance be found in [1].) The basic idea of any solution is probably to write (in meta-language):

```
Initial_operations;
Print_Twice(Initial_operations;
Close_off;);
Close_off;
```

Of course there is the problem that the procedure 'Print_twice' has to be defined, and its call printed, but that's a minor point ...

Here are two solutions to this problem in plain TEX, first one that prints itself, in typewriter type, on an otherwise blank page.

```
\output{}\def\do#1{\catcode`#112}\def\t{\dospecials\obeylines\tt~}
\def~#1^^:{#1#1^^:\end}\t
\output{}\def\do#1{\catcode`#112}\def\t{\dospecials\obeylines\tt~}
\def~#1^^:{#1#1^^:\end}\tz
```

The following solution is a variation on the original theme: it gives the source as message on the screen.

```
\catcode 13=12 \newlinechar 13
\def \a #1{\let \#\relax \let \a \relax \newlinechar 13\immediate \write 16{#1
\catcode `\#=12 \a {#1}}\end }
\catcode `\#=12 \a {\catcode 13=12 \newlinechar 13
\def \a #1{\let \#\relax \let \a \relax \newlinechar 13\immediate \write 16{#1
\catcode `\#=12 \a {#1}}\end }}
```

The reader may enjoy coming up with more variations, for instance a LATEX document that produces itself, or a plain TEX document that produces its LATEX source, or ...

References

[1] Douglas Hofstadter, Gödel, Escher, Bach, an eternal golden braid. New York 1979.

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

◇ Ron Sommeling
Centrum voor Wiskunde en Informatica
Kruislaan 413
1098 SJ Amsterdam
the Netherlands
ron@cwi.nl

A Font and a Style for Typesetting Chess using LaTeX or TeX

Piet Tutelaers

The Berkeley Font Catalogue [3] demonstrates how a chess font in combination with troff can be used to typeset chess diagrams. This article has inspired me to build a chess font with METAFONT from the nice font, see diagram 1, I once bought from Schaakhuis De Haan (Arnhem, The Netherlands). This 'font'

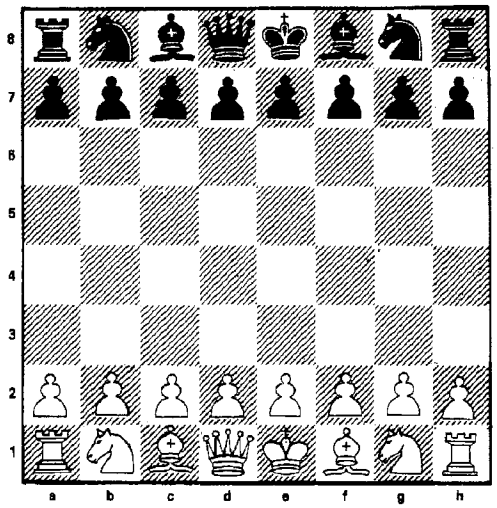


Diagram 1: Original font
(original size of board 9 × 9 cm)

consists of a set of chess boards and separate sets of chess pieces. The pieces have to be pasted on the board after pulling them from a sheet of paper. This nowadays is still common practice for publishers. Before I used METAFONT I made enlargements of the pieces on graph-paper using my stereo microscope for which I have a drawing prism. This graph-paper makes it easy to read the coordinates of points that need to be given to METAFONT. For the development of the font I have used AmigaMETAFONT which has graphical support. The design of the chess style has been done with AmigaTeX. Both programs run comfortably on my private Amiga1000 with 2.5 Megabytes of internal memory.

A chess font consists of 26 characters, with one character for the empty light square and one for the empty dark square. For each chess piece (Pawn, kNight, Bishop, Rook, Queen, King) there are four characters to represent that piece (White, Black) on both squares (light, dark). The troff chess font has also extra characters for the border of the board. These borders are added by the chess diagram macros as horizontal and vertical rules in my

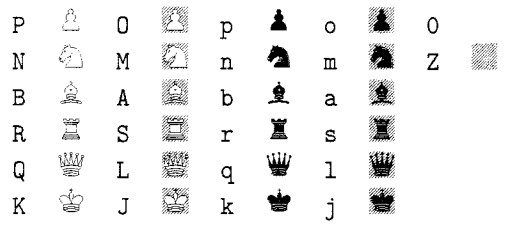


Table 1: Character encodings of `chess10`

approach. Table 1 shows all characters from font `chess10` (the size of a square being 10 points).

There is an extra font `chessf10` that contains only the so-called chess figurines (King, Queen, Rook, Bishop, kNight). With this font the move 25. N5×g3, in short algebraic notation, can be typeset as 25. ♘5×g3. In addition to `chess10` there are `chess20` and `chess30`. The 20-point version is used in the chess style because the diagrams made with it fit nicely in a twocolumn A4 page. But it would be easy to make another size font by changing only one parameter.

The king has given me the biggest trouble to METAfy. If you compare the original font with my METAimitation, you will see a few differences. For one thing, the chess board has no labels for rows and lines. These can be added to the diagram macros if desired. The pieces in the original font use shadings to get a better contrast with the dark squares. The rest of the differences have to be ascribed to my insufficient knowledge of METAFONT.

I have long hesitated to publish my METAcopies of the chess font in *TUGboat*. I have seriously tried to find the designer or owner of the original font. According to the Dutch firm that has taken over Schaakhuis De Haan, I could safely publish them because the fonts are not sold any longer. I hope this article helps in finding the designer of this very nice chess font and that he is not upset with my METAcopy of it.

Having a nice set of chess fonts is one thing; typesetting chess using them is another thing. When I accepted the editorship of our 'Schaakmaatje', as my chess club 'Schaakclub Geldrop' calls its chess magazine, I used TeX and some macros to typeset chess diagrams. After giving the LaTeX course at our Computer Center, I definitely wanted to move to this macro set. Especially the many available styles, and the need to have a simple macro for typesetting tables, makes LaTeX a lot simpler to use.

Chess playing macros

When some macros to play chess in TeX appeared in *TUGboat* [2], I used the ideas presented to make

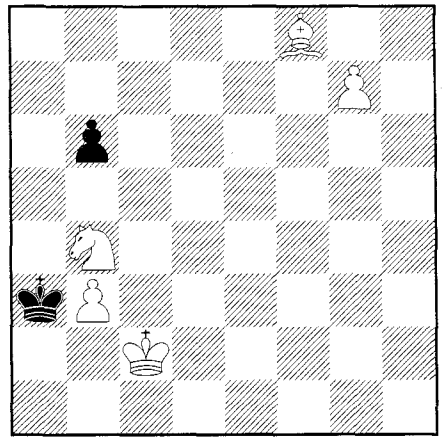


Diagram 2: White mates in three moves

my own chess style. As a typical example of an annotated chess game I have used a part of the game Fischer played against Tal during the Candidates' Tournament of 1959. In this tournament the sixteen year old Fischer lost four times from Tal who by winning this tournament earned the right to meet Botvinnik for the world championship. The game shows a real fight between two very offensive players. The game itself is annotated by Fischer. Both the input and the output are included on page 6.

To typeset the main line with automatic updating of the chess position, `chess.sty` has the macro `\newgame`, which starts a new game, and the environment `position`, to set up a position other than the initial one. The chess position after the 25th move in the game Fischer–Tal (see diagram 3 on page 6) is defined with:

```
\begin{position}
\White(Kh1,Qe6,Re1,a2,b2,c2,g2,h2)
\Black(Kf8,Qb8,Rd7,Rg8,Be7,a6,b4,h7)
\global\Whitetrue\global\movecount=25
\end{position}
```

`\Whitetrue` gives the turn to White; `\Whitefalse` gives it to Black. Setting the move counter is achieved with `\movecount=25`. The `\global` is needed because both commands are used inside an environment.

There exists another macro `\board` for defining a chess position in case automatic updating is not wanted. This macro is used for the mate in three problem (see diagram 2):

```
\board{ * * B *}
      {* * * P }
      { p * * *}
      {* * * * }
      { N * * *}
      {kP* * * }
      { *K* * *}
      {* * * * }
```

Notice that the user of `chess.sty` doesn't need to know the character encodings from table 1! He only needs to know the abbreviations of pieces (uppercase for White and lowercase for Black) and that empty squares are represented by a ␣ (light) or a `*` (dark).

To show the board in either case, one needs to call the macro `\showboard` or `$$\showboard$$` if the board should be centered.

To automatically update a position defined by the `position` environment there are two macros: `\ply` and `\move`. If White's move is not followed by some analysis, the macro `\move` can be used. Otherwise the move has to be broken down into two plies (half moves) with `\ply`. The argument(s) of `\ply` and `\move` contain the from square followed by the to square of the moving piece, or the King in case of castling. A square is represented as a column `[a-h]` followed by the row `[1-8]`. In correspondence chess a similar notation is used except for the columns which are also denoted as decimals `[1-8]`. I think that using letters is less confusing and reflects the way (at least in Europe) chess players think.

If a pawn arrives at its final destination, it becomes a Queen, Rook, Bishop or kNight `{Q|R|B|N}`. If the promotion piece is omitted a default Queen is taken. Moves can also be commented with things like `!` for good moves, `??` for exceptionally bad moves, and so on. So the syntax, in a free style of Extended Backus Naur Form, of an argument for both `\ply` and `\move` can be described as:

```
[a-h][1-8][a-h][1-8]{{Q|R|B|N}comment}
```

The translation of this move representation to long algebraic notation is carried out by the 'invisible' macro `\@ply`. For example: `\@ply g1f3` will result in ♘g1–f3 in case square f3 is empty or ♘g1×f3 in case of a capture, `\@ply e1c1` will result in 0–0–0.

To update and query the chess board, represented by 64 macros (`\a1`, `\a2`,... `\h8`), `\@ply` uses the private macros `\@set` and `\@get`. The value of a square can either be empty (letter `E`), a White piece (`Q`, `R`, `B`, `N`, `P`) or a Black piece (`q`, `r`, `b`, `n`, `p`). To update the chess position `\@ply g1f3` does a `\@set[g1](E)` to make the square g1 empty and a `\@set[f3](N)` to move the kNight to f3. The macro `\@ply` handles castling and the special pawn moves en passant capture and promotion. Because of its length the macro `\@ply` is not included. Basically, it is just a set of nested conditionals to cover all cases. Instead the macros `\@set` and `\@get`, that might be of interest to other TEX applications, are included here, together with `\@showchar` (the macro that maps a square value to the correct character

encoding using \@get) and \@emptyboard (a macro for clearing the chess board using \@set).

```
\def\@set[#1#2](#3){
  %arguments: [a-h1-8](<letter>)
  \expandafter
  \xdef\csname#1#2\endcsname{#3}}

\def\@get#1[#2#3]{%arguments:\cs[a-h1-8]
  \edef#1{\csname#2#3\endcsname}}

\newcount\@c   % column
\newcount\@r   % row
\newcount\@sum % row+column

\def\@col{\ifcase\@c\or a\or b\or c\or
  d\or e\or f\or g\or h\fi}
\def\@showchar{
  \@get\piece[\@col\the\@r]
  \if\piece E \ifodd\@sum 0\else Z\fi\else
  \if\piece P \ifodd\@sum P\else O\fi\else
  \if\piece p \ifodd\@sum p\else o\fi\else
  \if\piece R \ifodd\@sum R\else S\fi\else
  \if\piece B \ifodd\@sum B\else A\fi\else
  \if\piece N \ifodd\@sum N\else M\fi\else
  \if\piece r \ifodd\@sum r\else s\fi\else
  \if\piece b \ifodd\@sum b\else a\fi\else
  \if\piece n \ifodd\@sum n\else m\fi\else
  \if\piece K \ifodd\@sum K\else J\fi\else
  \if\piece Q \ifodd\@sum Q\else L\fi\else
  \if\piece k \ifodd\@sum k\else j\fi\else
  \if\piece q \ifodd\@sum q\else l\fi\else
  \fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi}

% inner loop needs extra { ... }

\def\@emptyboard{
   \@r=1
   \loop
      {\@c=1
       \loop
         \@set[\@col\the\@r](E)
         \ifnum\@c<8 \advance\@c by1
       \repeat}
       \ifnum\@r<8 \advance\@r by 1
   \repeat
   }
```

Analysis mode

In order to save space tournament bulletins often prefer short algebraic notation. In this notation the from square of a piece is omitted. If a move otherwise would be ambigious, because another piece can reach the same destination, the letter of the column or the number of the row from the originating square is added. The short and long algebraic notation is also used in analysis mode when we annotate moves from the main line or a variation. To make the typesetting of these move notations easy and compact, I have introduced a pair of |...| to activate the algebraic notation. Unfortunately this character pair cannot be used within arguments of macros. But you can of course use the pair outside the macro like `|\centerline{25. Kf8!}|`. Or if you prefer you can use the chess-environment instead. Because LaTeX uses this character in the tabular environment to draw vertical rules I have made an environment nochess that can be used to prevent the | character from activating the analysis mode. Within the nochess environment you can use the environment chess to get analysis mode again. Here follow some examples to show how short and long algebraic notation can be generated in analysis mode:

input	output
`|21. Nf3-e5??, 0-0!|`	21. ♘f3–e5??, 0–0!
`|21.: Ke8*f8|`	21. ..., ♔e8×f8
`|21. Nfe5, K*f8|`	21. ♘fe5, ♔×f8
`|21.: f8Q+|`	21. ..., f8♕+

In the analysis mode some characters have a special meaning. The '*' becomes a '×' to denote a capture, the '-' is mapped to '–' and the ':' is used as an abbreviation for ' ...,' as one needs if the analysis starts with a Black move.

The chess macros also have support for different languages thanks to the possibilities of the babel package from Johannes Braams [4]. This package can be used within plain TeX and LaTeX in TeX3.0 or in older versions. The names of the chess pieces can be chosen according to their Dutch abbreviations by specifying the language dutch before the chess style:

```
\documentstyle[dutch,chess]{...}
```

In Dutch the letters K ('koning'), D ('dame'), T ('toren'), L ('loper') and P ('paard') are used to denote the king, queen, rook, bishop and knight respectively. These letters are used in the macros \move and \ply, only if you provide a promotion piece, within the analysis mode and in the position environment. In the \board macro we use the same letters to denote the White pieces and the corresponding lowercase letters to indicate the Black piece. In this macro we need also a letter to represent the pawn. Because both 'pion' (pawn) and 'paard' (knight) start with a 'p' I have choosen the letter 'I' for the 'pIon'.

Hopefully the examples above and the extract of the game Fischer–Tal on page 6 are further self explaining.

Tournament style

To illustrate the tournament style I have included the complete example game of Fischer against Tal

	(LA)TEX	meaning
⩲	\wbetter	White stands slightly better
⩱	\bbetter	Black stands slightly better
±	\wupperhand	White has the upper hand
∓	\bupperhand	Black has the upper hand
+−	\wdecisive	White has a decisive advantage
−+	\bdecisive	Black has a decisive advantage
→	\withattack	with attack
⩳	\betteris	better is
○∞○	\seppawns	separated pawns

Table 2: Informant symbols

from [5] in the style that is used in the Chess Informant [7]. In this book special symbols are used for often used chess terminology. Table 2 shows the list of the symbols I needed in the Fischer game. In [5] the moves 50... ♔c7 51. ♖b5 are omitted. But this is corrected in his later book [6].

BOBBY FISCHER – MIKHAIL TAL

Belgrade (ct/27) 1959

1. e4 c5 2. ♘f3 d6 3. d4 cd4 4. ♘d4 ♘f6 5. ♘c3 a6 6. ♗c4 e6 7. ♗b3 [7. 0–0 ♗e7 8. ♗b3 ♕c7 9. f4 b5 10. f5 b4 11. fe6!? (11. ♘ce2 e5 12. ♘f3 ♗b7⩱) bc3 12. ef7 ♔f8 13 ♗g5 ♘g4!∓] **b5!** [7... ♗e7? 8. f4 0–0 9. ♕f3 ♕c7 10. f5! (10. 0–0? b5 11. f5 b4! 12. ♘a4 e5 13. ♘e2 ♗b7⩱) e5 11. ♘de2 b5 12. a3 ♗b7 13. g4±→] **8. f4!? b4! 9. ♘a4 ♘e4 10. 0–0 g6?** [⩳10... ♗b7] **11. f5! gf5** [11... ef5? 12. ♗d5 ♖a7 13. ♘f5! gf5 14. ♕d4] **12. ♘f5! ♖g8** [⩳12... d5 13. ♘h6 ♗h6 14. ♗h6; 12... ef5? 13. ♕d5 ♖a7 14. ♕d4] **13. ♗d5! ♖a7** [13... ed5 14. ♕d5 ♗f5 15. ♖f5 ♖a7 16. ♕e4 ♖e7 17. ♕b4 ♖e2 18. ♗g5! ♖g5 19. ♖g5 ♕g5 20. ♖b8+−PANOV] **14. ♗e4?** [⩳14. ♗e3! ♘c5 15. ♕h5! ♖g6 (15... ♘a4 16. ♗a7 ed5 17. ♖ae1) 16. ♖ae1! KEVITZ] **ef5 15. ♗f5 ♖e7! 16. ♗c8 ♕c8 17. ♗f4?** [⩳17. c3! ♕c6 18. ♖f2; 17. ♕d6? ♖g2 18. ♔g2 ♖e2 19. ♔f3 ♗d6 20. ♔e2 ♕c2−+] **♕c6! 18. ♕f3 ♕a4** [18... ♕f3 19. ♖f3 ♖e2 20. ♖f2 ♖f2 21. ♔f2⩲○∞] **19. ♗d6 ♕c6! 20. ♗b8 ♕b6** [20... ♕f3? 21. ♖f3 ♗g7 22. c3+−] **21. ♔h1 ♕b8 22. ♕c6** [22. ♖ae1 ♔d8! 23. ♖d1 ♔c7! (23... ♔c8? 24. ♕c6+−) 24. ♕f4 (24. ♖d4 ♕b7!) ♔b7 25. ♖d6 ♕c7 26. ♕b4 ♔c8 27. ♖a6 ♕b7! 28. ♕b7 ♔b7 29. ♖af6 ♖g7=; 22... ♖g6? 23. ♕f7 ♔d7 24. ♖d1! ♖d6 25. ♖d6 ♔d6 26. ♖f6!+−] **♖d7 23. ♖ae1** [23. ♖ad1 ♗d6 24. ♖f7 (24. ♖f6 ♖g6 25. ♖dd6 ♕d6!) ♕c7; 23. ♖f7 ♕d6] **♗e7** [23... ♔d8? 24. ♖f7! ♗e7 25. ♖fe7 ♖e7 26. ♖d1+−] **24. ♖f7 ♔f7 25. ♕e6 ♔f8!** [25... ♔g7? 26. ♕d7+−] **26. ♕d7** [26. ♖f1? ♔g7 27. ♖f7 ♔h8 (28. ♕d7 ♖d8 29. ♕g4 ♕e5−+)] **♕d6 27. ♕b7 ♖g6 28. c3! a5** [28... bc3 29. ♕c8 ♗d8 30. ♕c3=] **29. ♕c8** [⩳29. cb4! ♕b4 (29... ab4 30. a3! ba3 31. ba3 ♕a3=) 30. ♕f3 ♔g7 31. ♕e2=] **♔g7 30. ♕c4 ♗d8 31. cb4 ab4** [31... ♕b4 32. ♕e2=] **32. g3?** [32. ♕e4 (32... ♗c7 33. ♕e7 ♔g8 34. ♕e8 ♕f8 35. ♕e4)] **♕c6 33. ♖e4 ♕c4 34. ♖c4 ♖b6!** [34... ♗e7? 35. a3!] **35. ♔g2 ♗f6 36. ♔f3 ♔e5 37. ♔e3 ♗g5 38. ♔e2 ♔d5 39. ♔d3 ♗f6 40. ♖c2? ♗e5 41. ♖e2 ♖f6 42. ♖c2 ♖f3 43. ♔e2 ♖f7 44. ♔d3 ♗d4! 45. a3** [45. b3? ♖f3 46. ♔e2 ♖f2 47. ♕d3 ♖c2 48. ♔c2 ♔e5+−] **b3 46. ♖c8** [46. ♖e2 ♖f3 47. ♔d2 ♗b2; 46. ♖d2 ♖f3 47. ♔e2 ♖f2] **♗b2 47. ♖d8 ♔c6 48. ♖b8 ♖f3 49. ♔c4 ♖c3 50. ♔b4 ♔c7 51. ♖b5 ♗a1 52. a4 b2!** [53. ♔c3 b1♕!] **0 : 1 [Fischer]**

The LaTeX input of the above game looks like:

```
% Some macros to improve readability...
\newcommand{\finito}[2]{%
   {\bf\hfill#1\hfill[#2]\par}}
\newenvironment{mainline}[2]{\bf
   \newcommand{\result}{#1}%
   \newcommand{\commentator}{#2}%
   \begin{chess}}%
   {\end{chess}\finito{\result}%
    {\commentator}}
\newenvironment{variation}{[\begingroup
   \rm\ignorespaces}%
   {\endgroup]\ignorespaces}
...
\begin{mainline}{0~:~1}{Fischer}
1. e4  c5   2. Nf3 d6 3. d4  cd4
4. Nd4 Nf6  5. Nc3 a6 6. Bc4 e6
...
13. Bd5! Ra7
   \begin{variation}
   13: ed5 14. Qd5 Bf5 15. Rf5 Ra7
   16. Qe4 Re7  17. Qb4 Re2
   18. Bg5! Rg5 19. Rg5 Qg5
   20. Rb8\wdecisive
   \nochess PANOV \endnochess
   \end{variation}
14. Be4?
...
\end{mainline}
```

The mainline is typeset in boldface and the variations in roman. When we need to typeset a name we have to avoid the analysis mode by using the `nochess` environment. The `nochess` environment can be used in LaTeX style (`\begin{nochess}...\end{nochess}`) or in the shorter TeX style (`\nochess...\endnochess`).

Using plain TeX

To use `chess.sty` in plain TeX you need the following line for including the chess macros:

```
\input chess.sty
```

The `nochess` and `position` environments from `chess.sty` should be replaced by `\nochess` ... `\endnochess` and `\position` ... `\endposition` pairs.

Further wishes

The chess font and the chess style form a good basis for publishing about chess as is demonstrated by [1, 8]. But there are still some wishes to make the writing really enjoyable. My first wish would be a program with a chess board interface on which I can set up a position, play a variation and add text to the computer generated chess moves, go back to the main line, play another variation, and so on. I have seen an X11-based chess interface using hyperbuttons which provides a good starting point to make such a program!

Surely the chess style can be improved and other style conventions added. If anybody does so, please let he send me his improvements.

Availability

This article, the chess fonts and the corresponding style file with the complete game of Fischer against Tal and other examples can be retrieved (files `chess.tar.Z` and `chess.README`) from the file server `sol.cs.ruu.nl (131.211.80.5)` via anonymous ftp from the directory `TEX`.

Acknowledgements

I would like to thank Victor Eijkhout for his help and criticism he gave me to improve both the chess style and the readability of this article. My thanks go also to Hugo van der Wolf for polishing my English, and to the `UseNet` users who have sent me bug reports and have reported inconveniences present in version 1.0. Most of them are solved and will be made available in version 1.2.

References

[1] *Tournooimagazine van de Halve Finale Ned. Kampioenschap Schaken*, ed. by Anjo Anjowierden, Enschede. 1990.

[2] *Typesetting Chess*, by Wolfgang Appelt, *TUGboat* 9, no. 3, December 1988.

[3] *Berkeley Font Catalogue*, Ultrix–32 Supplementary Documents, Digital Equipment Corporation, Merrimack, New Hamshire, 1984.

[4] *Babel, a multilingual style-option system for use with LaTeX's standard document styles*, by Johannes Braams, *TUGboat* 12, no. 2, June 1991.

[5] *My 60 Memorable Games*, by Bobby Fischer, Faber and Faber, London. 1969. ISBN 0-571-09312-4

[6] *Fischer's Chess Games*, by Bobby Fischer, Oxford University Press, Oxford. 1980. ISBN 0-19-217566-1

[7] *Chess Informant 51*, ed. by Aleksandar Matanović, Šahovski Informator, Belgrade. 1991. ISBN 86-7297-024-1

[8] *Dragon, The Bulletin of the Cambridge University Chess Club*, ed. by Steve Rix and John Wilson, December 1991. Cambridge.

◇ Piet Tutelaers
Computer Center
Eindhoven University of Technology
P. O. Box 513
5600 MB Eindhoven, NL
internet: `rcpt@urc.tue.nl`

Example of the LaTeX-input and output of an annotated chess game using `chess.sty`

```
\begin{position}
\White(Kh1,Qe6,Re1,a2,b2,c2,g2,h2)
\Black(Kf8,Qb8,Rd7,Rg8,Be7,a6,b4,h7)
\global\Whitetrue\global\movecount=25
\end{position}
\begin{figure}
\centerline{Diagram~3: Fischer--Tal
 after 25.~\ldots, {\Fig K}f8!}
$$\showboard$$
\end{figure}

(See diagram~3.)

\ply  e6d7
Not |26. Rf1+, Kg7; 27. Rf7+, Kh8;
and if 28. Q*d7, Rd8; 29. Qg4, Qe5|
wins.

\ply       b8d6
\move d7b7 g8g6
Within a handful of moves the game
has changed its complexion. Now it
is White who must fight for a draw!

\ply  c2c3!
Black's extra piece means less with
each pawn that's exchanged.

\ply       a6a5
On |28.: b*c3; 29. Qc8+, Bd8;
30. Q*c3|=.

\ply  b7c8+
On the wrong track. Right is
|29. c*b4!, Q*b4 (if 29.: a*b4;
30. a3!, b*a3; 31. b*a3, Q*a3 draws);
30. Qf3+, Kg7; 31. Qe2| draws, since
Black can't possibly build up a
winning K-side attack and his own
king is too exposed.

\ply       f8g7
\move c8c4 e7d8
\move c3b4 a5b4
On |31.: Q*b4; 32. Qe2| White should
draw with best play.
$$\showboard$$
```

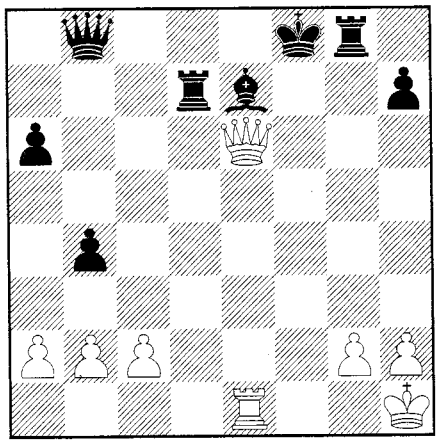
(See diagram 3.)

26. ♕e6×d7

Not 26. ♖f1+, ♔g7; 27. ♖f7+, ♔h8; and if 28. ♕×d7, ♖d8; 29. ♕g4, ♕e5 wins.

26. ... ♕b8–d6

Diagram 3: Fischer–Tal after 25. ..., ♔f8!



27. ♕d7–b7 ♖g8–g6

Within a handful of moves the game has changed its complexion. Now it is White who must fight for a draw!

28. c2–c3!

Black's extra piece means less with each pawn that's exchanged.

28. ... a6–a5

On 28.... b×c3; 29. ♕c8+, ♗d8; 30. ♕×c3=.

29. ♕b7–c8+

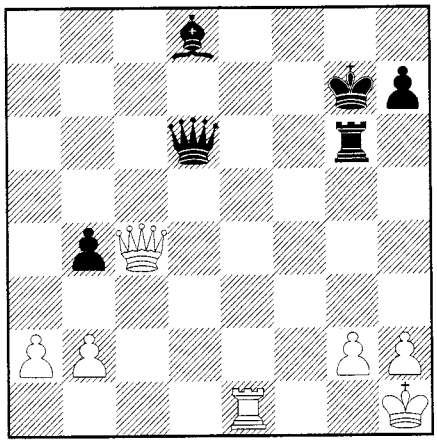
On the wrong track. Right is 29. c×b4!, ♕×b4 (if 29.... a×b4; 30. a3!, b×a3; 31. b×a3, ♕×a3 draws); 30. ♕f3+, ♔g7; 31. ♕e2 draws, since Black can't possibly build up a winning K-side attack and his own king is too exposed.

29. ... ♔f8–g7

30. ♕c8–c4 ♗e7–d8

31. c3×b4 a5×b4

On 31.... ♕×b4; 32. ♕e2 White should draw with best play.



Tower of Hanoi, revisited

Kees van der Laan

Abstract

Another version of programming 'The Tower of Hanoi' in TEX is provided.[1] No nodding knowledge of Lisp is required; just plain TEX. There is no restriction on the number of disks, apart from the installed limits of TEX. Generalized disks can be moved as well.

Introduction

At the Dedham TUG91 conference, I attended David Salomon's advanced TEX course. Instead of redoing his clear and ample exercises I decided to rework Leban (1985), the more so because elaborating a classic example might bring you to fundamental issues. In courses on programmming, the Tower of Hanoi problem is used to illustrate paradigms. I was pleased to encounter some paradigms of TEX programming while revisiting the tower.

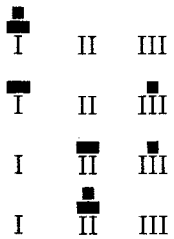
The Tower of Hanoi problem

A pyramid of disks—meaning a tower with implicit ordering of the disks—has to be moved under the restrictions that only one disk at a time can be moved and that each intermediate state consists of pyramids, obeying the original implicit ordering. In total three places for (intermediate) pyramids are allowed. For a pyramid of n disks, the solution needs $2^n - 1$ moves. For an introduction to the problem see the first paragraphs in Graham et al. (1989).

Example

```
\input hanoi.tex \hanoi2 \bye
```

will yield the process of replacements for 2 disks from tower I to tower II



[1] For an earlier article on the issue, see Leban (1985).

The file hanoi.tex

This file contains all the macros: the top `\hanoi`, the version with more parameters `\Hanoi`, the macro for the moves `\movedisk`, along with the auxiliary macro to prefix a string, `\lopoffx...\lopoffx...`, and the auxiliaries for printing, `\showtowers` and `\pt`.

As data structure a simplified version of Knuth's list, see TEXbook Appendix D.2, is used. A tower has the replacement text $\backslash\backslash\langle item_1\rangle\ \backslash\backslash\langle item_2\rangle \ldots \backslash\backslash\langle item_n\rangle$, where in this case each item is a control sequence. The separators, `\\`, "...are enormously useful, because we can define `\\` to be any desired one-argument macro and then we can *execute* the list!"

```
%hanoi.tex version 19 dec 91
\newcount\n   %The number of disks
\newcount\brd %Breadth of towers
\newcount\hgt %Height of maximum tower
\newcount\dskhgt %Height of each disk
\let\ea=\expandafter %Shorthand
\let\ag=\aftergroup  %Shorthand
\def\preloop{%To create loopcnt, a
             %local loopcounter
             %(see also loopy.TeX).
   \bgroup \advance\count10 by 1
   \countdef\loopcnt=\count10
            %Symbolic name
   \loopcnt=1 %(default)
 }%end \preloop
\def\postloop{\loopcnt=0 %Restore
   \egroup}%end \postloop
%
%Hanoi macros, top level
\def\hanoi#1{%Argument can be digit(s)
             %or a counter (numeric)
 \n=#1 %Assign argument value to \n
 \def\II{}\def\III{}%Empty towers
 %Next is inspired by the TeXbook,
 %p374, 378
 %The initial tower for \I is created
 %   \def\I{\\\i\\\ii\\\iii...\\\'n'}
 %next to the defs for \i,\ii,...\'n'.
 \preloop\ag\def\ag\I\ag{%
  \loop
    \ea\xdef\csname\romannumeral\loopcnt
            \endcsname{\the\loopcnt}
    \ag\\%separator
    \ea\ag\csname
         \romannumeral\loopcnt\endcsname
    \ifnum\loopcnt<\n
    \advance\loopcnt by 1
  \repeat   \ag}
```

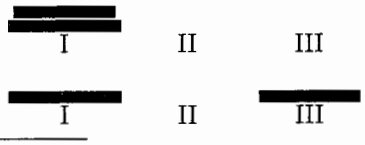
```
 \postloop
 %For printing, values are needed for
 \brd=\n %Breadth of largest disk
 \advance\brd by 3 %Little room extra
 \dskhgt=1 %Height of disks
 \hgt=\n\multiply\hgt by2 %\hgt is height
       \advance\hgt by1   %of towers
 \showtowers %Print initial state
 \Hanoi\I\II\III\n
 }%end \hanoi
%
\def\Hanoi#1#2#3#4{%Moves from #1 to #2,
                   %with aid of tower #3.
 %The number of disks is #4, in a counter.
 \ifnum#4=1 %For Tower of 1 disk,
            %just move the disk
   \movedisk\from#1\to#2%
   \showtowers%Print towers after move
 \else%Problem of #4 disks is solved by
      %- problem of (#4-1) disks,
      %- a move, and
      %- a problem of (#4-1) disks.
   {\advance#4 by-1 \Hanoi#1#3#2#4}%
   \movedisk\from#1\to#2%
   \showtowers%Print towers after move
   {\advance#4 by-1 \Hanoi#3#2#1#4}%
 \fi}%end \Hanoi
%
%Moving of the disks, TeXbook, App. D.2
%Slightly adapted versions of \lop (
%called \movedisk with function that
%first element of #1 is prefixed to #2)
%and \lopoff modification
\def\movedisk\from#1\to#2{%Move disk from
                   %tower #1 to tower #2
  \ea\lopoffx#1\lopoffx#1#2}
\def\lopoffx\\#1#2\lopoffx#3#4{\ea\gdef%
  \ea#4\ea{\ea\\\ea#1#4}
  \gdef#3{#2}%restore stub}%end\lopoffx
 }%end \movedisk
%
%Printing tower status
\def\showtowers{%Display pyramids
 \par\quad\hbox{\pt\I\ \pt\II\ \pt\III
              }\par
 }%end \showtowers
%
%Auxiliaries
\def\gobble#1{}%To eat character
%
\def\\#1{\hbox to\brd ex{\hss
  \vrule width#1ex height\dskhgt ex
                          \hss}%
 }%end \\
%
\def\pt#1{%Print Tower.
 %#1 is \I, \II, or \III
 \vbox to\hgt ex{\baselineskip=.2ex\vss
      #1%
      %Format pointer underneath
      \hbox to\brd ex{\hss
             \ea\gobble\string#1\hss}%
      }%end vbox
 }%end \pt
```

Disks not restricted to one digit

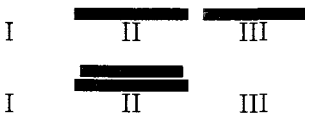
One could invoke `\hanoi{10}`[2] at the expense of ample time and use of paper. In order to illustrate the possibility of the macros to cope properly with disks denoted by more than one digit, the pyramid $\begin{matrix}/\,9\,\backslash\\ /\,10\,\backslash\end{matrix}$ can be handled via

```
\n=2    %Number of disks
\def\ix{9}\def\x{10}%Disks
\def\I{\\\ix\\\x}\def\II{}\def\III{}
\brd=10 %Breadth of largest disk
\hgt= 6 %Height of tower
\dskhgt=1 %Height of disks
\def\\#1{\hbox to\brd ex{\hss
  \vrule width#1ex height\dskhgt ex
                          \hss}}%
\showtowers%Initial state
\Hanoi\I\II\III\n
```

with result



[2] Or an invocation with an even larger argument. I allowed my modestly equipped MS-DOS PC to loop for $\backslash n = 1, 2, 3, \ldots$ in order to experience TEX's limits with respect to the practical upper bound for the size of the tower. A tower of 10 disks did take ≈1.5 minutes to TEX, with empty `\showtowers`. I had the patience to await the TEXing up to the tower of size 16, and concluded that there are no limits for reasonable usage. (Note that every increase of the size by 1 will double the time needed.) So, I don't know when my PC will crash or, to paraphrase the monks, 'when the world will end.' I also modified the macros into versions with `\I`, `\II`, and `\III` defined as token variables. I was surprised to see *that* version run 4 times slower. Whatever the value, the hanoi macros can be used as a program to compare the efficiency of TEX implementations.



Generalized disks

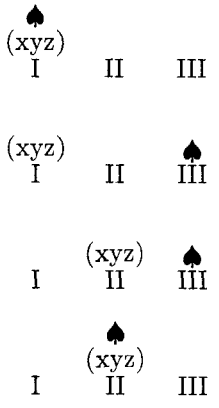
What about for example (xyz) as disk? Let us assume for printing that the contents of the pyramids—the strings—will do, in the implicit provided order. This can be obtained via a modified \\ definition.

Example

The tower can be moved, with the states printed via the following.

```
\n=2\def\i{$\spadesuit$}\def\xyz{(xyz)}
\def\I{\\\i\\\xyz}\def\II{}\def\III{}
\brd=6 \hgt=7
\def\\#1{\hbox to \brd ex{\hss
                          #1\hss}}%
\showtowers%Initial state
\Hanoi\I\II\III\n
```

with results



Interactivity

Downes (1991) inspired me to think about direct communication with the user. What about the modification of \showtowers into an appropriate \immediate\write16{...} command, such that the moves will appear on the screen? No previewing or printing![3]

```
%Direct screen \showtowers
\def\showtowers{
 \immediate\write16{
  \ea\gobble\string\I:   \I
  \ea\gobble\string\II:  \II
  \ea\gobble\string\III: \III}}
```

Conclusion

Is this just for fun? It was appropriate for 'Fun with TEX,' NTG's 91 fall meeting at Eindhoven. Furthermore, I experienced the following fundamental (TEX) programming issues

- recursion in solving the problem
- the use of the list data structure and separators to execute the list; see the TEXbook Appendix D.2
- creating and using a local loop counter[4]
- creation of a dynamic number of command names and a string of dynamic length
- generalizing the problem (not only numbers can denote disks; note that no comparison of disks is done, the ordering of the subtowers is maintained implicitly)
- direct communication on screen.

TEXnically \aftergroup, \countdef, \csname, \expandafter, \ifnum, \ifx, \immediate\write16..., \loop, \romannumeral, and \string are exercised.

The hardest thing was to get the towers aligned when formatting commands were split over several lines, due to the two-column format. Several % symbols were needed to annihilate the effect of spurious spaces, especially those created by some ⟨cr⟩'s.

It did take some time to realize the benefits of Knuth's list macros, not to say that I wandered around quite a bit.

References

[1] Downes, M.J. (1991): Dialogue with TEX. Proceedings TUG91.

[2] Graham, R.L, D.E. Knuth, O. Pastashnik (1989): Concrete Mathematics. Addison-Wesley.

[3] Knuth, D.E. (1984): The TEXbook. Addison-Wesley.

[4] Leban, B. (1985): A solution to the Tower of Hanoi problem using TEX. *TUGboat* 6, no. 3, 151 – 154.

[3] Alas, no control over the format on the screen either.

[4] This was mentioned before by Pittman (1988). It is a matter of taste and programming style whether one prefers this next best to the hidden counter idea, above the use of a global counter, for counting the number of times a loop is traversed. The difference in efficiency is negligible.

(Review. Leban requires as input the number of disks and the first tower as a sequence of one-digit numbers. This means that creation of the initial tower is not part of the paper. Furthermore, Leban develops a set of Lisp-like functions for TEX. Apparently Knuth's list macros, TEXbook Appendix D.2, have been overlooked. Leban's small list processing system is an example of reinventing the wheel, with the concept of an active list separator absent. As a consequence the printing of the towers is done by recursion which is not necessary when using Knuth's list separator appropriately to 'execute' the list.)

[5] Pittman, J.E. (1988): Loopy.TeX. *TUGboat* 9, no. 3, 289–291.

[6] Salomon, D. (priv. comm.)

◇ Kees van der Laan
Hunzeweg 57, 9893PB
Garnwerd (Gr), The Netherlands
cgl@rug.nl

LATEX

The LATEX Column

Jackie Damrau

Earlier Column Revisited

Claudio Beccari from the Institute of Technology of Turin, Italy, sent some interesting addenda and an answer to a question that appeared in the LATEX column in *TUGboat* 11, no. 4. Without further ado, here are Claudio Beccari's comments.

1. The minipage footnote counter is named `mpfootnote`, not `footnote`. If you want minipage footnotes numbered with lower case roman numerals, you can change the appearance of the minipage footnote counter with the following command:

```
\renewcommand{\thempfootnote}%
   {\roman{mpfootnote}}
```

   In any case, the footnote marks are typeset as exponents in math mode so that the math italic font is used by default. If you want the regular roman font, or any other font, you have to request it explicitly:

```
\renewcommand{\thempfootnote}%
    {{\rm\alph{mpfootnote}}}
```

2. The double spacing scheme suggested by J. Colmenares does not work if the commands — `\single` and `\double` — are issued while in `\normalsize`. In fact, `\baselinestretch` operates only when the size is changed, but `\normalsize` does not actually change size if the size is already normal. In order to have these commands work, you need to type: `\small\single` or `\small\double` .

   By doing this, the `\normalsize` command embedded in `\single` and in `\double` acts as it should with the new value of `\baselinestretch` becoming active.

   A better command, that does not require changes to size in advance and operates also when in `\normalsize`, unfortunately requires the category recoding of `@`. The macro appears below:

```
\newcommand{\single}%
    {\leadingstretch{1.0}}
\newcommand{\double}%
    {\leadingstretch{1.5}}
\makeatletter
\newcommand{\leadingstretch}[1]%
    {\let\@tempa\@currsize
     \let\@currsize\empty
     \def\baselinestretch{#1}\@tempa}
\makeatother
```

Footnotes: Problems and Solutions

Sometimes you need to attach footnotes to entries in tables. If you enclose a `tabular` environment within a `minipage` environment within a `table` environment, your footnotes float around together with your table.

But if you have to tag two or more table entries with the same footnote number (or letter) while you have one footnote text, you cannot use `\footnotemark[`⟨*number*⟩`]` several times in the table and `\footnotetext[`⟨*number*⟩`]{`*text*`}` after the `\end{tabular}` and before the `\end{minipage}` commands.

In fact if you do so, you will notice that the footnote text will be tagged with a number while the table entries are tagged with letters (default).

There is no simple means to go around this problem (at least I did not find a simple way out) while preserving the feature devised by L. Lamport of separating the footnote marks from the footnote texts, as he explains on pages 99 and 156 of the LaTeX book.

1st solution

Explicitly place multiple footnote marks as math exponents in the table entries. Afterwards typeset the footnote texts using only `\footnotetext[`⟨*number*⟩`]{`*text*`}` with its optional argument that agrees with the exponents that were set.

2nd solution

- Redefine a new boolean variable, say `tablenote`:

  ```
  % \tablenote is false by default
  \newif\iftablenote
  ```

- Redefine `\table` so that it sets `\tablenotetrue`.
- Redefine the `\footnotemark` and `\@xfootnotemark` commands so they operate on the `mpfootnote` counter, instead of `footnote`, if `tablenote` is `true`.
- Tag all your table entries that required tagging with the same mark (except the first one, which is marked with the full `\footnote` command) with the `\footnotemark[`⟨*number*⟩`]` that makes use of its optional argument.

3rd solution

Define a new environment. Locally redefine `\c@footnote` and `\thefootnote` to be equivalent to `\c@mpfootnote` and `\thempfootnote` respectively, using `\let`.

I used the first two solutions. The second one is definitely better, but it requires that you know where you put your hands within the internal LaTeX macros. The third solution seems very simple.

Maybe someone has an even better solution?

Claudio BECCARI
Department of Electronics
Institute of Technology of Turin, Italy
Corso Duca degli Abruzzi 24
I10129 – TORINO, Italy
E-MAIL: beccari@itopoli.bitnet

◇ Jackie Damrau
SSC Laboratory
Mail Stop 1011
2550 Beckleymeade Avenue
Dallas, TX
email: damrau@sscvx1.ssc.gov

Errata: "See also" indexing with Makeindex

Harold Thimbleby

In *TUGboat* 12, no. 2 (page 290) I gave the LaTeX definitions to enable an author to obtain 'see also' entries in their index. I am grateful to Professor John C. Slattery of Texas A&M University for pointing out that they did not work.

The following correction works for me (using *Textures* and LaTeX 2.09), but not for Slattery who is using a NeXT, though the same version of LaTeX:

```
\def\subsee#1#2{{\em see also\/} #1}
%       the #2 consumes a comma
\def\nosee#1{}
%       consume the page number
\def\seealso#1#2{\index
  {#1!zzzzz@\string\subsee{#2}|nosee}}
```

The intention is, given the definitions as shown above, and supposing index entries for "Scheme" (`\index{Scheme}`) occur on pages 147 and 401, this is how `\seealso{Scheme}{LISP}` would end up in the index:

> Scheme, 147, 401
> *see also* LISP

If you have the problems reported by Slattery, `\seealso` must written out in full with you manually replacing the parameters `#1` and `#2` with what you want.

I made two errors in the original note: First, I published a fragment of LaTeX without testing it exactly as it appeared in print. The second error was conceptual. I naïvely forgot that a TeX definition is referentially opaque: I had assumed that given `\def\seealso{x}`, then `\seealso` can be written for `x` (with the exception of various cases where `x` contains things like `\futurelet`). In my case I had checked `x` but not the form `\seealso` that I used in the article. I had been fooled by the innocent appearance of `\index{argument}` — and I had not appreciated the LaTeX manual's remark that `\index` should not appear inside another command's argument, as it does here with `\def`.

I apologize for inconvenience caused, and I will look forward to any suggestions for a general solution. Is there any way for macros like LaTeX's `\index` to detect when they are being used improperly?

◇ Harold Thimbleby
Stirling University
Stirling
Scotland, FK9 4LA

LaTeX 2.09 ↪ LaTeX3

Frank Mittelbach and Chris Rowley

Abstract

This is a brief sketch of the LaTeX3 Project: its history, its present state and its future, as at the end of 1991. It is based on a talk given by Frank Mittelbach, technical director of the LaTeX3 Project, in November 1991 to a meeting of the Nordic TeX Users Group; the handout from this talk was recently published in TeXline [Mit92].

1 The LaTeX3 Project

The LaTeX3 Project was initiated at the Stanford annual meeting in August 1989 but the first, somewhat vague, plans had already been formulated in 1988 when Rainer Schöpf and Frank, after sending several pages of bug fixes for LaTeX 2.09 to Leslie Lamport, received a positive answer.

1.1 History

Whenever the future is somewhat unpredictable it seems wise to take a look into history — to find out what has already been achieved and what remains to be tackled.

From the now quite long history of the LaTeX3 Project we can observe a growing bulk of syntax descriptions and (partial) implementations that are the results of three years work.

Looking at the original goals for a reimplementation, described in the Stanford paper [MS89], it would appear that nearly everything has now been achieved.

- The NFSS, which is now in widespread use, provides a very general font selection method. The extended syntax, on which we are now working, also provides for scaled fonts.
- With `amstex.sty` the mathematical capabilities of LaTeX have reached (at least) the standard of AMSTeX.
- With the new implementation by Denys Duchier (this supersedes `array.sty`), together with valuable suggestions by several others, tabular processing will have reached a very high standard.
- A new error recovery and help system is being developed. This should provide a safe and easy-to-learn environment for novice users.
- A more flexible input language is being developed, in which environments and commands can have attributes specified. This will also allow easy conversion from SGML to LaTeX3, for suitable DTDs.

Milestones:
Syntax and implementation

1988 – Some bug fixes sent to Leslie Lamport
– Four page sketch of NFSS
1989 – First implementation of NFSS
– Implementation of `amstex.sty`
1990 – New tabular implementation by Denys Duchier
– First attribute prototype
– First kernel prototype
– First recovery/help prototype
1991 – Second kernel prototype
– Sketches for style designer interface
– Second description of the attribute concept
– Extended description of the help facility
– Syntax for extended NFSS
– Third kernel prototype
– Release of LaTeX 2.09 international with NFSS support

How did all this happen?

All the work has been carried out in the free time of several individuals and involves, as you can imagine, a lot of enthusiasm to keep the project alive. So far, more than thirty people have contributed in one way or another to the effort.

One of the major, and growing, problems is how to bring people from all over the world together to discuss the open questions and find new solutions. It is important that these meetings involve people from outside the project since we very much need the views and experience of typesetters, designers, publishers, etc. to help eliminate the flaws in the system and find new and better solutions.

In this regard both the London and the Dedham workshops were hugely successful, but further workshops of this kind are essential if we are to provide LaTeX3 with a suitable designer interface.

1.2 Current state

So why are we now saying that this project is still only at its start? Because we have learnt that our

Milestones:
Meetings, Workshops and Correspondence

1988 – Non-flame answer from Leslie Lamport
1989 – Talk in Stanford
– Meetings with Leslie in Stanford
– Talks in Karlsruhe
1990 – Mega-bytes of e-mail correspondence
– Working week in Mainz with Leslie
– Talk in Cork
1991 – More mega-bytes of e-mail correspondence
– Workshop in London
– Meeting with Leslie in London
– Workshop in Dedham
– Working week in Providence with Michael Downes
– Working week in Mainz

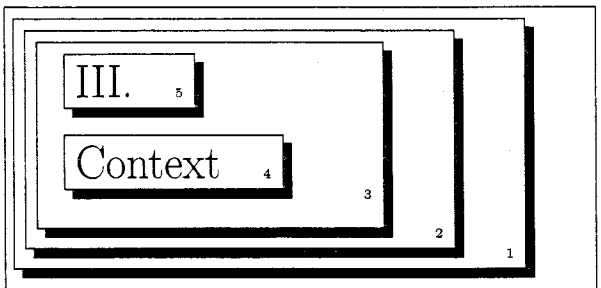
original goals did not touch many of the real problems as we see them today — there has thus been a major change of focus.

Working on the project has led us to the conclusion that one gains very little by just providing more and more specialized style files to solve this or that special problem. Instead, we think that, in order to provide a fully flexible and extensible system, the major effort in the future must go into the design of the right style interface — one that allows easy implementation of various layouts. (Easy, of course, is relative: easy compared to the complexity of the task.)

The project plan which emerged from this change of focus can be summarized as follows.

- The development of a new internal language that is more suited to the expression of visual components of the layout process.
- The development of high-level generic functions that allow the straightforward expression of most commonly used layout components.
- The development of a model for specifying and modifying all of the parameters that influence the layout.

Since the syntax for this internal language is still changing on a daily basis, and the generic functions depend on it, in this article we shall concentrate on the *model* for parameter setting rather than its detailed syntax.



1. document context
2. section context
3. heading context
4. title context
5. number context

2 The model for parameter setting

The new system will implement a model of document formatting based on the fact that the formatting required for any particular part of the document depends on its context. Therefore the setting of layout parameters must be context-sensitive.

2.1 The concept of a context

What do we mean by a 'context'?

The nesting of the entities in a document is clearly important for the specification of its layout. For example, in LaTeX2.09 a second-level list (i.e. a list nested within another list) will be treated, by the standard styles, in a different way from a first-level list.

The order (or sequencing) of entities is also important. For example, the space after a section head will probably depend on what comes next: ordinary text or another, lower-level, head.

However, the context of an entity in a document is not given simply by the nesting and sequencing of its surrounding entities. Take, for example, what happens when some footnote or float that appears in a list itself contains a list. Because of the incorrect handling of contexts in the current LaTeX the inner list is typeset as a second-level list. In other words, an entity must be able to (partially) forget about its context or, more generally, must be able to manipulate its context.

Moreover, the context of an entity also has a 'visual' component — this is its relationship to the rest of the document *after* it is formatted. This 'visual context' usually depends on the formatting of many other entities in the document.

For example, the optional argument to `\marginpar` enables the author to specify that the

text of a marginal should depend on its visual context, i.e. on whether it finally appears on a recto or a verso. Thus its formatting depends, at least, on the formatting of all the entities which appear on nearby pages.

To summarize, at every point in the document we are in a logical context given by the nesting and sequencing of all the entities in the document; and we are also in a visual context determined by decisions made about the formatting of all the entities in the document.

2.2 Contexts in LaTeX3

Therefore the new system will, as far as possible, allow the settings of all the parameters (needed to specify the formatting of an entity) to vary according to the current context, both logical and visual. Here the idea of 'parameter' is used in a very general way — e.g., the particular generic function used to format some entity is considered to be a parameter so that, via this concept, different generic functions can be used in different contexts.

For example, it will be possible to change the layout of certain entities within floats by specifying the values of their layout-parameters in the context of "floats" differently from those applied in the main text.

- In this article, lists within floats are indented at both the left and the right margin (we had to do this the hard way for LaTeX2.09).
- A table entity in a float can be set in a smaller typesize than one in the main text.

Another case where the nesting of entities may affect the formatting is a list within a tabular p-entry: such a list may well be typeset 'in-line', i.e. a new item does not start a new line.

It will also be possible to specify formatting parameters dependent on the sequencing of entities. This is needed for design specifications such as: "The vertical space after a theorem is 3pt if it is immediately followed by a proof, otherwise it is 6pt." Another example is the specification of no indentation when a paragraph immediately follows a certain entity, e.g. an article title.

Visual-context-dependent specifications which will be implementable in LaTeX3 include cases where the formatting depends on page-breaks or column-breaks; where it depends on the type of page; and where it depends on the types of objects appearing on a page. For example: "A section head should be preceded by a text-width rule and have 5 lines of text after it before a page-break. If it appears at the top of a page then the rule is omitted."

2.3 Some problems

Implementing such a model within the TeX framework poses some interesting challenges.

1. The user input is not (in SGML jargon) a 'normalized document' — i.e. it may contain entities which are hidden inside user-defined shorthands so that they cannot be easily prescanned.
2. The specification of contexts in terms of the sequencing of entities is important but its efficient implementation is restricted somewhat by the underlying TeX engine.
3. Taking the visual component of contexts into account requires the use of a multi-pass system, and even then it is, in practice, restricted by the formatter's capabilities.

Let us look at the first of these problems in more detail. The problem here is that the document (`.tex`) file is not a normalized form of the document (one in which every entity is explicitly tagged) thus we cannot, before we start typesetting an entity, scan forward using TeX's native scanner to see what 'begin' or 'end' tags are coming up. This means that certain formatting decisions have to be taken without knowing what follows. Possible solutions to this problem are

- forbid the use of user-defined shorthands
- do the scanning 'by hand' using TeX's macro language
- use a two-pass system that normalizes the document in the first pass
- use a multi-pass system in which each pass uses information on sequencing gathered during the previous pass.

The first of these would be unacceptable as it would remove one of the strengths of LaTeX, its flexibility. The second would vastly increase processing time since TeX's native scanner is highly optimized. The latter two do not seem to be feasible for a system that uses TeX as the input language, but this should be explored further.

With respect to the second problem, TeX has no built-in mechanism to detect whether simple character material is about to be contributed to some horizontal list immediately after a given tag. Only after the material has been contributed to the horizontal list can one deduce this fact by 'dirty tricks' involving special kerns. Thus this can be used only for interrupting a context sequence — the already contributed material cannot be manipulated further.

The last problem is of a more theoretical nature. All of the currently available automated-typesetting

systems format each document entity in a predetermined visual context, i.e. they assume that the visual context can be determined completely by the logical nesting and sequencing of entities. To a certain extent TEX is an exception as it applies dynamic programming to the process of paragraph formatting, and this involves the recomputation of contexts for ligatures etc.; the use of this kind of process needs to be extended much further in order to produce automated typesetting of high quality.

For example, in TEX there is very little interaction between the paragraph-building mechanism and the page-building mechanism. Thus the way in which TEX avoids a hyphen at the end of a page (a good example of the visual context affecting the formatting of a word) is by moving the whole line instead of by rejustifying the paragraph to avoid a hyphen at that point. These matters are discussed further by the authors in [MR92a].

This type of consideration draws the boundary between the ideal model and the real world. It also reveals that certain optimizations in the TEX language necessarily restrict the flexibility of high-level front-ends built on it. (Since TEX is a Turing machine, it is possible to move this boundary but not without an unacceptable increase in processing time.)

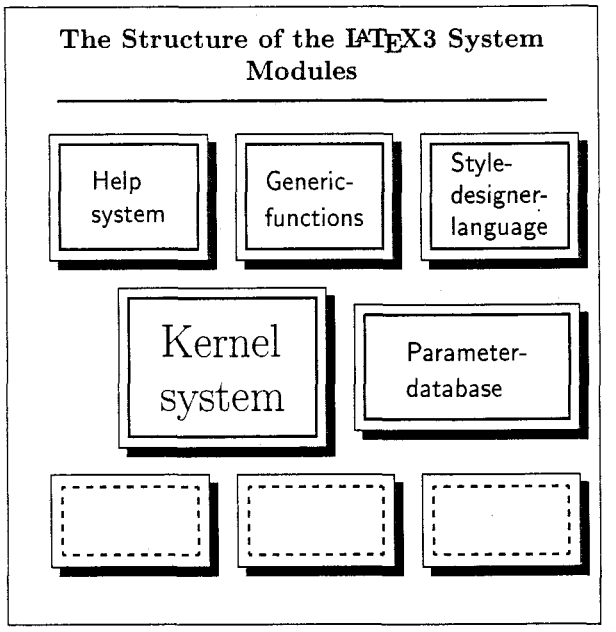
So let us now turn to an overview of the major components of the LATEX3 system.

3 The structure of the system

The LATEX3 system contains many interwoven structures each of which could be the subject of a separate article. The kernel of the system provides the basic data structures such as lists, stacks, etc., used to program higher-level modules. It will also contain arithmetic functions for integers and dimensions, e.g., it will be possible to express relationships between individual parameters by specifying assignments that contain expressions.

On top of it are built the generic formatting functions and manipulation functions for the parameter database allowing context-sensitive parameter specification. Together these will form the basis for the style designer language.

One other important component of the system will be an interactive help system allowing extensive help texts as well as the possibility of defining system reactions dependent on user actions. Help messages and such additional error-correcting code will be held in external files that are read in when an error is detected by the system. In this way an elaborate, interactive help and error-handling mechanism will be possible whilst keeping the LATEX3 kernel compact.



It is important to distinguish between document styles — these are written in the style designer language (and contain no, or nearly no, TEX code in the traditional sense) — and additional modules, some of which will provide additional functionality for use in a range of document styles (e.g., facilities for version control and change bars, enhanced graphical features, etc.) whilst others will define extra entities for typesetting specialized classes of documents (e.g., functions for critical text editions, for specialized math constructions, chemical formulas).

The new kernel will provide a programmer interface that makes the development of further such modules a reasonable task.

4 The new generation of LATEX users?

Over the last six years, the way in which TEX and its front-ends are commonly used has changed dramatically. Here is a quotation from some unknown novice of LATEX.

"Dear Sir,

I have successfully installed LATEX from the distribution — the file `sample` was just printed. However, somewhere in the README files a similar program called TEX is mentioned. Could you please explain to me how to install this program? ..."

From a phone call.

This might sound funny at first, but this extreme is not far from the reality for many users (and even support personnel) nowadays.

- Today the overwhelming majority of users use LaTeX only.
- Today there are very few users who have more than a minor understanding of the underlying system, and they usually have no knowledge of the TeXbook — even of its existence.
- Today most users can be nicely classified as "has heard of 'macros', but has never seen one".

This does not mean that today's users are less intelligent than past users, but simply that they are using a plug-and-play TeX system.

Nowadays TeX, and LaTeX as its major front-end, have to compete with the so-called Desktop Publishing Systems. To keep them alive we have to bridge the gap between the 'implementor/wizard' type of user of the '80s and the new type who uses the system just as one tool out of many, with no understanding of its internals.

With the LaTeX3 Project we hope to achieve this goal.

5 Establishing contact with the project

There is now a public list for discussion of the design of LaTeX3. This list is *not* for questions about, or discussion of, how to solve problems within the current LaTeX — so please refrain from asking questions such as "How do I avoid getting double-spaced tabulars whilst using `doublespace.sty`?" But, of course, it is OK to point out problems with the current LaTeX which should be addressed by the new version. We must stress this request not to misuse the list, otherwise there will be no new LaTeX before 2001!

We are now at the stage where we can identify sub-projects: these will vary in type and size, and will not all require advanced TeXnical expertise. Anyone seriously interested in finding out more about such (voluntary) work should contact one of the project managers.

The TUG office is administering a fund to help with the expenses of the project: further details of this can be found in [MR91], or [MR92b].

Contacts

As at February 29th 1992, the e-mail addresses of the LaTeX3 Project group managers are:

Frank Mittelbach
`Mittelbach@mzdmza.zdv.Uni-Mainz.de`
Chris Rowley
`C.A.Rowley@vax.acs.open.ac.uk`
Rainer Schöpf
`Schoepf@sc.ZIB-Berlin.de`

To subscribe to the LaTeX3 design list, send the following in an e-mail message to
`LISTSERV@dhdurz1.bitnet`:

```
SUB LATEX-L  Firstname Secondname
```

References

Further information about the project can be found in the articles listed below. This bibliography also contains entries concerning BibTeX, which will be reimplemented and enhanced by Oren Patashnik for use with LaTeX3.

[Gue91] Mary Guenther, editor. *TeX 90 Conference Proceedings*, March 1991. Published as *TUGboat* 12(1).

[Mit90] Frank Mittelbach. LaTeX 2.10. In Lincoln K. Durst, editor, *1990 Conference Proceedings*, page 444, September 1990. Published as *TUGboat* 11(3).

[Mit92] Frank Mittelbach. LaTeX2.09→LaTeX3. *TeXline*, (14):15–18, February 1992.

[MR91] Frank Mittelbach and Chris Rowley. The LaTeX3 project fund. *Die TeXnische Komödie*, 3(4):13–15, December 1991.

[MR92a] Frank Mittelbach and Chris Rowley. The pursuit of quality — How can automated typesetting achieve the highest standards of craft typography? In C. Vanoirbeek and G. Coray, editors, *Electronic Publishing '92*, Cambridge University Press, April 1992.

[MR92b] Frank Mittelbach and Chris Rowley. The LaTeX3 project fund. *TeX and TUG News*, 1(1):5–6, February 1992.

[MS89] Frank Mittelbach and Rainer Schöpf. With LaTeX into the nineties. In Christina Thiele, editor, *1989 Conference Proceedings*, pages 681–690, December 1989. Published as *TUGboat* 10(4).

[MS90a] Frank Mittelbach and Rainer Schöpf. LaTeX dans les années 90. *Cahiers GUTenberg*, (6):2–14, July 1990.

[MS90b] Frank Mittelbach and Rainer Schöpf. LaTeX limitations and how to get around them. In Brüggemann-Klein, editor. *1989 EuroTeX Conference Proceedings*. To appear.

[MS90c] Frank Mittelbach and Rainer Schöpf. Reprint (with corrections): The new font family selection — user interface to standard LaTeX. *TUGboat* 11(2):297–305, June 1990.

[MS90d] Frank Mittelbach and Rainer Schöpf. Towards LaTeX 3.0. *TEX Gebruikers Group*, (2):49–54, September 1990. Reprint of [MS91].

[MS91] Frank Mittelbach and Rainer Schöpf. Towards LaTeX 3.0. In Guenther [Gue91], pages 74–79. Published as *TUGboat* 12(1).

[Rhe90a] David Rhead. Could LaTeX do more for chemists? *TEXline*, (12):2–4, December 1990. Suggestions for LaTeX3.

[Rhe90b] David Rhead. Towards BibTeX style-files that implement principal standards. *TEXline*, (10):2–8, May 1990.

[Rhe91] David Rhead. How might LaTeX3 deal with citations and reference lists? *TEXline*, (13):13–20, September 1991. Suggestions for LaTeX3.

[WM91] Reinhard Wonneberger and Frank Mittelbach. BibTeX reconsidered. In Guenther [Gue91], pages 111–124. Published as *TUGboat* 12(1).

◇ Frank Mittelbach
EDS, Electronic Data Systems
(Deutschland) GmbH
Eisenstraße 56 (N15)
D-6090 Rüsselsheim
Federal Republic of Germany
Internet:
mittelbach@
mzdmza.zdv.uni-mainz.de

◇ Chris Rowley
Open University
Walton Hall
Milton Keynes MK7 6AA
United Kingdom
Internet:
c.a.rowley@vax.acs.open.ac.uk

Les Cahiers GUTenberg
Contents of Recent Issues

Numéro 9 - July 1991

Henry THOMAS, Typographie du jeu d'échecs; pp. 66–73

This article presents different existing solutions for typesetting chess, either in LaTeX, PostScript or with a Macintosh.

Announcement of the First European Summer School in Digital Typography, EPFL, Lausanne, September 23–29, 1991; p. 74

André HECK, Star-TEX; pp. 75–78

TEX and LaTeX have been used from the beginning and are still extensively used by astronomers and space scientists around the world for their mail, for writing papers, for putting together newsletters, proceedings, reports, books, and so on. Some publishers have also set up their own sets of macros for journals and/or book series. It seems however that the tendency is presently to pull out of TEX and go towards more user-friendly and performant systems. A meeting on Desktop Publishing in Astronomy and Space Sciences will take place at Strasbourg Astronomical Observatory (France) in October 1991.

Yannis HARALAMBOUS, Quand TEX rencontre Mozart...; pp. 79–81

This is a report on the 10th annual meeting of DANTE, the association of German-speaking TEX users, which took place in Vienna, 20–22 February, in this Mozart anniversary year.

Announcement for *Premiers pas de LaTeX*, a French adaptation by Éric Cornelis of a manual by Michael URBAN; p. 82

Bernard GAULLE, L'association ... fait la force; pp. 83–85

The recent events in the (LA)TEX world, like the work on fonts with 256 characters or the first book published by GUTenberg or lastly the preparation of the 6th European TEX Conference, readily demonstrate the usefulness and benefit of the association of people working as a team. This article ends with a call for new volunteers.

Éric PICHERAL, Distribution MlTEX v.3.1.4 pour Sun; pp. 86–87

An announcement and description of the distribution, adapted for French use on Sun 3 and Sun 4, of MlTEX, related software, and fonts.

Announcement of EP92 : Electronic Publishing 92, EPFL, Lausanne, 7–10 April 1992; p. 87

6ème conférence TEX européenne/GUTenberg'91; pp. 88–90

An announcement of these two meetings to be held in Paris, 23–26 September 1991.

Numéro 10–11 - September 1991
Proceedings of the 6th European TEX Conference and GUTenberg'91

The 6th European TEX Conference took place in Paris on 23–25 September 1991, followed by GUTenberg'91 on 26 September. Papers in these Proceedings were presented at the TEX conference unless otherwise noted. In the case of multiple authors, the presenter of a paper is indicated by an asterisk.

Basil MALYSHEV, Alexander SAMARIN*, and Dimitri VULIS, Russian TEX; pp. 1–6

This article presents the TEX extension for processing russian texts. Russian TEX is based on version 3.0 and virtual fonts. Different coding schemes for russian characters are allowed.

Theo JURRIENS, TEXniques in Siberia; pp. 7–13

This article summarizes the problems of giving a LaTeX course in Siberia. It concludes with an overview concerning the future of TEX inside the USSR.

Announcement for *Premiers pas de LaTeX*, a French adaptation by Éric Cornelis of a manual by Michael URBAN; p. 14

Jörg KNAPPEN, TEX and Africa; pp. 15–24

At the present time, TEX is not usable for typesetting many african languages. They use special letters which don't occur in the standard fonts (and aren't included in the ec-scheme). The letters used in the major languages of Africa can be put into *one* font. A font encoding scheme (fc) and some METAFONT code have been prepared. There is work in progress on hausa TEX (by Gos Ekhaguere, Ibadan).

Oussama BOUGHABA*, Seifeddine BOUTALBI et Michel FANTON*, Vers une version arabisée de TEX; pp. 25–44

This paper presents the state of development of an arabicized version of TEX for DOS.

Basil MALYSHEV and Alexander SAMARIN*, TEX Integrated Shell for IBM PC; pp. 45–55

This article presents the TEX Integrated Shell (TIS) — a special environment for TEX on the IBM PC to conceal some problems from an ordinary user. TIS contains the screen interface for different actions during TEXing. It can be configured to satisfy a user's requirements and hardware & software conditions. It downloads only the files to be used, in particular, pixel font files which are required for a given `.dvi` file.

Infothèque — la librairie de l'informatique (advertisement); p. 56

Jiří ZLATUŠKA, Automatic generation of virtual fonts with accented letters for TEX; pp. 57–68

This paper presents an approach towards deriving fonts with accented letters for European languages using virtual fonts as an alternative to the development of genuine new fonts with METAFONT. The ACCENTS processor is presented as a tool for mechanization of the process by enabling automatic generation of accented font layout and the virtual font definition from the TFM file of the source font in the TEX text encoding, and from an auxiliary input containing corrections of accent placement for specific characters.

Yannis HARALAMBOUS, ScholarTEX; pp. 69–70 (abstract only)

ScholarTEX is a software package consisting of fonts, TEX macros, executables (for the Macintosh), and a detailed manual with examples and exercises. ScholarTEX allows easy and efficient use of TEX for typesetting in many languages.

[Editor's note: For technical reasons, the text of this article could not be published in the Proceedings, and will appear in a later issue of the *Cahiers*.]

Johannes BRAAMS, Babel, a multilingual style-option system; pp. 71–72 (abstract only)

The babel system of style-option files adapts LATEX (and plain TEX) to a multi-lingual environment. This paper presents a summary of the ways in which 'hardwired' use of the english language has been 'repaired'. Language-dependent typographical conventions are examined using examples from the publications of the European Community. Some of the problems arising in processing documents using more than one language (for example, more than one set of hyphenation patterns) are discussed.

[Editor's note: Only an extended abstract appears in the Proceedings; for further information, readers should refer to "Babel, a multilingual style-option system for use with LATEX's standard document styles" in *Nederlandstalige TEX Gebruikersgroep*, Verlag 6^e bijeenkomst, 91.1 (1991), pp. 75–83.]

Michel FANTON, TEX : les limites du multilinguisme; pp. 73–79

This paper describes the specific features of arabic typesetting and gives an account of the price to pay in developing an arabicized version of TEX.

IBM RISC System/6000 (advertisement); p. 80

Joachim SCHROD, An International Version of *MakeIndex*; pp. 81–90

MakeIndex is a powerful, portable index processor which may be used with several formatters. But it is only usable for English texts; non-English texts — especially with non-Latin alphabets, like Russian, Arabic, or Chinese — may not easily be worked on. The tagging of index entries is often tedious and error prone: If a markup is used within the index key, an explicit sort key must be given. A new version of *MakeIndex* is presented which allows the automatic creation of sort keys from index keys by user-specified mappings. This new version does support documents in non-Latin alphabets. Furthermore it needs less main memory than the former one, and may now be used for large indexes even on small computers.

Paul BACSICH, Ethel HEYES, Paul LAFRERE and Geoff YARWOOD, Conversion of Microsoft Word into LATEX; p. 91 (abstract only)

We describe a program which converts Microsoft Rich Text Format files (as produced by several word processing packages) into standard LATEX. This program converts character glyphs, character attributes, style information, fonts, lists and tables to their "equivalents" (if any) in LATEX.

Conversion of character, font and attribute information is hardwired in, to a set of tables in the program, which can be changed (by a programmer).

Conversion of style information is controlled by a Conversion Control File which can be amended by the user. This file assigns LATEX constructs to Word style tags.

The latest version of the program converts mathematical mark-up in Word Formula Mode to the LATEX equivalents. There are many difficulties with this approach and the paper will cover the main ones including the basic difficulty of recognizing in a word processor file what is mathematics and what is not.

The program has been used to convert a complete textbook, *Introduction to Information Technology*, by Dr P. Zorkoczy (Pitman, 1990), from Word into LATEX for use in a hypermedia system.

[Editor's note: The authors were unable to attend, and this paper was not presented.]

Trois vérités sur TEX (advertisement, Northlake Software); p. 92

Michel LAVAUD, AsTEX: an integrated and customizable multiwindow environment for scientific research; pp. 93–116

AsTEX is a program that runs on a PC under the control of Framework 3, and transforms it into an integrated and customizable multiwindow environment for scientific reasearch, as comfortable to use as the one of a workstation. It has been devised as a help to create scientific books and, more generally, as a help for everyday scientific work. It includes a hypertext-like file manager which allows to classify and archive all the files related to the current document by means of a hierarchy of explicit titles, and to retrieve any of them very easily, whetever its physical location. It allows also to display the structure of a LATEX document of any length, and to modify and restructure it in a completely interactive manner. It offers an interface with a local or distant Fortran compiler, which allows to perform numerical compilations from a LATEX document. It has also an interface with the computer algebra program Maple, to perform formal computations interactively from a text, a worksheet or a database, when the PC is connected to a Unix station through a LAN or through a modem.

A set of PCs equipped with AsTEX and connected by a LAN to a workstation can provide a low-cost alternative to a network of workstations, for laboratories and educational institutions already equipped with PCs, and that cannot afford or do not want to equip each researcher or student with a workstation.

Steen LARSEN* and Arne Flemming JENSEN, Tailored database publishing with TEX; pp. 117–134

TEX is well suited to produce inventories such as bibliographies or dictionaries. Such publications are characterized by a large number of entries, a high uniformity of structure, typographical variation, and high demands to line and page breaking. Furthermore, sorting of entries and compiling of indexes will often be necessary. In the necessary TEX input files there will be a large percentage of control sequences.

Producing inventories based on text editors presents numerous difficulties as regards, for example, TEX syntax control, data validation, and sorting. Producing them via a standard database system gives better data control, but forces the user to accept the limitations of the system's user interface.

This paper presents an approach chosen when establishing a tailored TEX-based database publishing system for the bibliography Nordic Archaeological Abstracts. The solution was implemented by combining three different systems: an interface management system, a database management system, and TEX. The system is described and compared to the previous editor-based production, and future possibilities are briefly sketched.

Bernard LEGUY, Drawing tree structures with GWEZ; pp. 135–146

GWEZ is a set of macros able to build tree structures and to draw them; these macros are written with TEX; they use only plain TEX commands and fonts and can as well be used with LATEX.

Kees VAN DER LAAN, Math into BLUes: sing your song; pp. 147–170

TEXing mathscripts is not simply typing. Math has to be translated into TEX commands. First the motivation for this work is given. Next traditional math page make-up is summarized along with the macroscopic math TEX commands. After answering 'Why TEXing mathscripts is difficult?' an anthology of TEX falls and their antidotes is discussed. At the end, suggestions are given in order to lessen the difficulties.

[Editor's note: The first part of this paper was published in *TUGboat* 12, no. 3, Proceedings of the TUG 1991 annual meeting. With the author's agreement, only the second part of the full paper appears in the Proceedings of the European TEX Conference. The first part, called *Mourning*, can also be found in *Nederlandstalige TEX Gebruikersgroep*, Verlag 6e bijeenkomst, 91.1 (1991), pp. 57–74.]

Angelika BINDING, Organizing a large collection of stylefiles; pp. 171–184

Springer Verlag has to maintain a large collection of macro packages for different layouts, for which there are versions both for plain TEX and LATEX and for different sets of fonts. We therefore designed a concept of modularising these packages and have implemented mechanisms to create formatfiles loading our individual set of fonts without changing the standard formatfiles `plain` and `lplain`.

Andrew E. DOBROWOLSKI, Typesetting SGML documents using TEX; pp. 185–196

Since its publication as an international standard in 1986, the Standard Generalized Markup

Language (SGML) has become a preferred document markup standard within many industries. Many users have developed their own document type definitions (DTDs) which define the elements (tag sets) for their documents. However, if SGML is to become a universally accepted standard of document interchange, then a standard way to specify formatted output and a means of producing that output will be needed.

The U.S. government's Computer-aided Acquisition and Logistic Support (CALS) initiative selected SGML as the standard of text interchange. The output specification section of the CALS standards proposed the Formatted Output Specification Instance (FOSI) as the means of formatted output specification interchange.

TEX can be used as the formatting engine to implement FOSI-based formatting. But without extending TEX not every FOSI formatting request can be fulfilled. Conversely, certain TEX capabilities cannot be formulated in terms of FOSI characteristics. However a FOSI/TEX based formatting system would be a major advance towards fulfilling the document interchange needs of a growing community of SGML users.

[Editor's note: This paper was first published in *TUGboat* 12, no. 3, Proceedings of the TUG 1991 annual meeting.]

Christophe CÉRIN, Vers la construction de macros de mise en couleur pour TEX; pp. 197–206, plus one color plate

This article presents a step-by-step approach to putting colour in TEX documents.

Bernd SCHMID, WYSIWYG-TEX-editors on the basis of object-oriented system technology; p. 207 (abstract only)

After a short introduction into object-oriented programming introducing the terms object, object attributes and methods, and after showing the motivation to realize a TEX-editor on the basis of object-oriented technology, the objective of the development of a WYSIWYG-editor and its range concerning TEX which is implemented is described.

The general strategy of realization will then be explained. For this the scanner-/parser implementation as well as the box concept and the box attributes will be described. This tends to demonstrate the easy, efficient interactive treatment of documents using WYSIWYG-suitable editing of TEX-terms and the reduction of mistakes by syntax-/semantics-checks using graphic methods of visualization. An outlook on further developments on the basis of this object-oriented concept of realization will be given.

Finally, the application of a WYSIWYG-editor is evaluated in the project "COMPINDAS-GUT" of Fachinformationszentrum Karlsruhe. This will include a depiction of the demands of this application and the extent of the project, a classification of the users as well as the evaluation of first experiences with the use of a WYSIWYG-editor concerning efficiency, user acceptance, and error reduction in comparison to current TEX editing tools.

Philippe LOUARN, Lucida, une fonte complète pour LATEX, et son installation; p. 208 (abstract only)

[Editor's note: This article was published in *Les Cahiers GUTenberg*, numéro 9, July 1991, pp. 32–40; see above for abstract.]

Maurice LAUGIER, Composition des formules chimiques en TEX; pp. 209–221

Formatting chemical formulae with TEX needs some special macros to describe links and ramifications. The authors describe their macros and present some illustrations.

[Editor's note: This paper was presented at GUTenberg'91.]

Calendar

1992

Apr 7 ***TUGboat* Volume 13, 2nd regular issue:** Deadline for receipt of news items, reports.

Apr 7–10 EP'92 Swiss Federal Institute of Technology, Lausanne, Switzerland. For information, contact ep92@eldi.epfl.ch.

May 18–22 Intensive LaTeX, Lexington, Massachusetts.

May 22 Polska Grupa Użytkowników Systemu TeX (GUST), first general meeting, Warsaw, Poland. For information, contact Hanna Kołodziejska (gust@alfa.camk.edu.pl).

May 23 CyrTUG: First Annual Meeting, Institute of High Energy Physics, Protvino (suburb of Moscow), Russia. For information, contact Irina Makhovaya (irina@mir.msk.su).

Jun 4 NTG Spring Meeting, "Scientific Publishing and TeX", CWI, Amsterdam, The Netherlands. For information, contact Gerard van Nes (vannes@ECN.NL).

Jun 15 Journée LaTeX Version 3, Paris, France. For information, write to GUTenberg, journée LaTeX3, BP 21, 78354 Jouy-en-Josas, France.

Jun 15–19 NTG Advanced TeX course, "Insights & Hindsights", Groningen, The Netherlands. For information, contact Gerard van Nes (vannes@ECN.NL).

Jun 16–18 [TeCH'92, a GUTenberg conference, has been canceled.]

Jun 17–22 Society for Scholarly Publishing, 14th Annual Meeting, "Information Encounters of the Scholarly Kind", Hyatt Regency, Chicago, Illinois. For information, contact SSP (301-422-3914).

Jun 18 GUTenberg journée, Geneva, Switzerland. Details TBA.

Jun/Jul ukTeXug: "Design Issues": A visit to the Dept. of Typography, University of Reading. For information, contact Peter Abbott (pabbott@nsfnet-relay.ac.uk).

TUG'92 Conference, Portland, Oregon

Jul 20–24 Intensive Beginning/Intermed. TeX

Jul 26 TeX for Publishers

Jul 27–30 **TUG Annual Meeting: "TeX in Context"**, Portland, Oregon. For information, contact the TUG office. (See page opposite inside back cover.)

Jul 31–Aug 1 Practical SGML and TeX Introduction to Typography LaMS-TeX

Aug 3–7 Advanced TeX and Macro Writing Intensive LaTeX

Aug 18 ***TUGboat* Volume 13, 3rd regular issue:** Deadline for receipt of *technical* manuscripts.

Sep 14–16 EuroTeX 92, Prague, Czechoslovakia. For information, contact Jiří Veselý (jvesely@cspguk11.bitnet). (See page 107.)

Sep 15 ***TUGboat* Volume 13, 3rd regular issue:** Deadline for receipt of news items, reports.

Oct 19–23 Beginning/Intermediate TeX, Chicago, Illinois.

Oct 26–30 Intensive LaTeX, San Diego, California.

Nov 19 NTG Fall Meeting, [topic to be announced], Meppel, The Netherlands. For information, contact Gerard van Nes (vannes@ECN.NL).

Nov 2–6: Beginning/Intermediate TeX, San Diego, California.

Status as of 6 April 1992

Nov	9–13	Intensive LaTeX, Providence, Rhode Island.
Nov	17	***TUGboat* Volume 14, 1st regular issue:** Deadline for receipt of *technical* manuscripts (tentative).
Dec	15	***TUGboat* Volume 14, 1st regular issue:** Deadline for receipt of news items, reports (tentative).

1993

Feb	16	***TUGboat* Volume 14, 2nd regular issue:** Deadline for receipt of *technical* manuscripts (tentative).
Mar	16	***TUGboat* Volume 14, 2nd regular issue:** Deadline for receipt of news items, reports (tentative).
Aug	17	***TUGboat* Volume 14, 4th regular issue:** Deadline for receipt of *technical* manuscripts (tentative).
Sep	14	***TUGboat* Volume 14, 4th regular issue:** Deadline for receipt of news items, reports (tentative).

For additional information on the events listed above, contact the TUG office (401-751-7760, email: `tug@math.ams.com`) unless otherwise noted.

EuroTeX 92 in Prague 14–18 September 1992

EuroTeX 92 is being organized by CSTUG, the organization for Czech- and Slovak-speaking TeX users, in collaboration with Charles University and Czech Technical University under the auspices of both Rectors. It will take place in Prague on 14–18 September 1992. (Please make a note in your diary.)

Accommodations and activities

Accommodations will be offered in the student hostel Kajetanka, a relatively modern facilty. Plans are being made to transfer participants each morning by bus from the hostel to Czech Technical University where the program will be held. (This would be a 25-minute walk from Katejanka.) Lunch and dinner will be served at the conference site.

There are plans for an opening party on Monday evening, a concert in a historical hall typical of Prague on Wednesday, and some other events. For those arriving Monday morning, a sightseeing tour may be offered on Monday afternoon. To keep things within everybody's budget, the main program is being offered as a package: the complete program from Monday to Friday morning (accommodation in double rooms, full pension from Tuesday to Thursday, opening party on Monday evening, concert, breakfast on Friday, conference fee and proceedings) for about 300 DM. Those who prefer a single room would pay an extra 60 DM. (Payment details, account numbers, etc., will be sent to registrants later.) For persons accompanying participants, a special program will be organized including visits to galleries and other places of interest. For an *additional* 50 DM a day, a limited number of participants may stay on till Sunday (one or two days more) either for tutorials or just to enjoy meeting friends and good beer in some of the pubs Good Soldier Svejk would visit.

Why come to EuroTeX 92?

In addition to invited talks by leading specialists you will have a chance to listen to lectures on different TeX applications and meet TeX friends from many countries. The meeting is the first to offer really extensive contacts with people from former East European countries ("from behind the iron curtain" — can you still remember that?). It takes place in the Golden Heart of Europe — Prague, one of the nicest capitals in Europe. You can visit it for a surprisingly low price: since we would like to make EuroTeX 92 in Prague accessible to the majority of TeX fans from all over Europe, we are arranging it on a modest but good level.

The date for submitting papers will be past by the time you read this. However, these are a few of the topics that have been proposed:

- Quo vadis, TeX?
- National versions and standardization
- Non-standard applications

- The use of TEX for small/newly emerging enterprises

Committee

Peter Abbott	Erich Neuwirth
Jacques André	Petr Novak
Jana Chlebikova	Stefan Porubsky
Bernard Gaulle	Philip Taylor
Karel Horak	Jiří Veselý
Joachim Lammarsch	Jiří Zlatuška

How to register

Those who are interested in taking part are kindly asked to fill in the following form and send it via e-mail to the address

`eurotex at cspguk11.bitnet`

I intend to take part in EuroTEX 92, Prague (14.-18.9.1992)

name:

userid:

node:

Please send more detailed information on EuroTEX to the following e-mail address (please fill in):

Or copy the following form, fill in using block letters, and send to the address:

CS TEX — EUROTEX 92
Mathematical Institute
Sokolovska 83
186 00 PRAHA 8 - Karlin
Czechoslovakia

Name: _____

Institution: _____

Position: _____

Address (business or home - circle one):

_____

_____

_____

_____

I intend to visit EuroTEX 92 in Prague (September 14. - 18.(20.), 1992)

_____

(Signature)

On behalf of the organizers:

Karel Horak
Jiří Veselý
Jiří Zlatuška

Production Notes

Barbara Beeton

Input and input processing

Electronic input for articles in this issue was received by mail and on diskette. Most articles were fully tagged for *TUGboat*, using either the `plain`-based or LATEX conventions described in the Authors' Guide (see *TUGboat* 10, no. 3, pages 378–385). Several authors requested copies of the macros (which we were happy to provide); however, the macros have also been installed at `labrea.stanford.edu` and other good archives, and an author retrieving them from an archive will most likely get faster service. Of course, the TUG office will provide copies of the macros on diskette to authors who have no electronic access.

The number of articles in this issue was heavily weighted toward LATEX. Of the major articles, only seven were *not* in LATEX; and of those, one was submitted as a `dvi` file. Pages were more evenly divided, with about 60% in LATEX. In organizing the issue, attention was given to grouping bunches of `plain` or LATEX articles, to yield the smallest number of separate typesetter runs, and the least amount of handwork pasting together partial pages. This also affected the articles written or tagged by the staff, as the conventions of `tugboat.sty` or `ltugboat.sty` would be chosen depending on what conventions were used in the preceding and following articles; no article was changed from one to the other, however, regardless of convenience.

We ran into only a few problems arising from one author's macro definitions affecting later articles, and these were relatively easily eliminated, once they were recognized. We did uncover another lapse in the LATEX routines for combining articles — failure to reset the figure counter resulted in the number of the first figure in one article to have a value considerably greater than 1.

Continuing with tradition, several articles required font work; these included the articles by Mylonas and Whitney (p. 39) and Tutelaers (p. 85).

The latter also required the use of PostScript output devices, and occasioned my first direct use of the Math Society's Compugraphic 9600 Imagesetter.

The article by Graham Asher (p. 13) was prepared using Type & Set, and delivered via the network as a uuencoded `dvi` file.

The following articles were prepared using the `plain`-based `tugboat.sty`:

- Toby Thain, *Packing* METAFONT*s into PostScript*, page 36.
- C. Mylonas and R. Whitney, *Modern Greek with adjunct fonts*, page 39.
- Alan Hoenig, *Just plain Q&A*, page 60.
- L. Siebenmann, *Elementary text processing and parsing*, page 62.
- Jonathan Fine, *Some basic control macros*, page 75.
- Kees van der Laan, *Tower of Hanoi*, page 91.
- abstracts of the *Cahiers GUTenberg*, page 101.
- the TUG calendar, page 106.
- announcement of EuroTEX 92 in Prague, page 107.
- these Production notes
- "Coming next issue"

Output

The bulk of this issue was prepared at the American Mathematical Society from files installed on a VAX 6320 (VMS) and TEX'ed on a server running under Unix on a Solbourne workstation. The Mylonas/Whitney article was prepared by Ron Whitney on an IBM PC-compatible 386 using PC TEX. The article by Asher was prepared on an IBM PC-compatible using EMTEX and the suite of pre- and post-processors described in the article. Most output was typeset on an APS-μ5 at the AMS using resident CM fonts and additional downloadable fonts for special purposes. The two exceptions were the articles by Rahtz (p. 34) and Tutelaers, which contained PostScript inclusions and were set on the Compugraphic 9600 Imagesetter at AMS.

One photograph, photographically screened in the traditional manner, appears in the article about Sam Whidden (p. 11).

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

Arrows for technical diagrams

David Salomon, requiring arrows of more varieties than are available in unextended (LA)TEX, has created a font of arrowheads. Since TEX does not have diagonal rules, only horizontal and vertical arrowheads were developed. However, the methods used can easily be extended for diagonal arrowheads. (Originally intended for the present issue, this item was delayed by technical problems in generating new fonts.)

Approaching SGML from TEX; SGML Questions and Answers

Two articles. The first, by Reinhard Wonneberger, present arguments for taking a standardized approach to document preparation using structural markup, as embodied in SGML, and for implementing the formatting with TEX.

The second, by Wonneberger and Frank Mittelbach, provides explanations of the terminology and concepts of SGML in a question and answer format.

ZzTEX: A macro package for books

Paul Anagnostopoulos describes the design decisions behind a macro package intended to produce books to varying specifications with a minimum of macro modification. A book is considered as a structure of blocks, each of which may contain independent design specifications as well as specs governing the interaction of adjacent or nested blocks. All the usual features of scientific ans scholarly books are supported, including cross-referencing and indexing.

XBibTEX and Friends

Support facilities to make BIBTEX input more straightforward and reliable are described by Chris Bischof.

Book reviews

New books about TEX and related subjects are appearing from every direction imaginable. Reviews of *LATEX for Everyone*, by Jane Hahn, and *Practical SGML*, by Eric van Herwijnen, are on hand, and at least one more is expected.

Institutional Members

The Aerospace Corporation,
El Segundo, California

Air Force Institute of Technology,
Wright-Patterson AFB, Ohio

American Mathematical Society,
Providence, Rhode Island

ArborText, Inc.,
Ann Arbor, Michigan

ASCII Corporation,
Tokyo, Japan

Beckman Instruments,
Diagnostic Systems Group,
Brea, California

Belgrade University,
Faculty of Mathematics,
Belgrade, Yugoslavia

Brookhaven National Laboratory,
Upton, New York

CERN, *Geneva, Switzerland*

Brown University,
Providence, Rhode Island

California Institute of Technology,
Pasadena, California

Calvin College,
Grand Rapids, Michigan

Carleton University,
Ottawa, Ontario, Canada

Centre Inter-Régional de
Calcul Électronique, CNRS,
Orsay, France

College of William & Mary,
Department of Computer Science,
Williamsburg, Virginia

Communications
Security Establishment,
Department of National Defence,
Ottawa, Ontario, Canada

Construcciones Aeronauticas, S.A.,
CAE-Division de Proyectos,
Madrid, Spain

DECUS, Electronic Publishing
Special Interest Group,
Marlboro, Massachusetts

Department of National Defence,
Ottawa, Ontario, Canada

Digital Equipment Corporation,
Nashua, New Hampshire

E. S. Ingenieres Industriales,
Sevilla, Spain

Edinboro University
of Pennsylvania,
Edinboro, Pennsylvania

Elsevier Science Publishers B.V.,
Amsterdam, The Netherlands

European Southern Observatory,
Garching bei München,
Federal Republic of Germany

Fermi National Accelerator
Laboratory, *Batavia, Illinois*

Florida State University,
Supercomputer Computations
Research, *Tallahassee, Florida*

Fordham University,
Bronx, New York

General Motors
Research Laboratories,
Warren, Michigan

Grinnell College,
Computer Services,
Grinnell, Iowa

Grumman Aerospace,
Melbourne Systems Division,
Melbourne, Australia

GTE Laboratories,
Waltham, Massachusetts

Hughes Aircraft Company,
Space Communications Division,
Los Angeles, California

Hungarian Academy of Sciences,
Computer and Automation
Institute, *Budapest, Hungary*

IBM Corporation,
Scientific Center,
Palo Alto, California

Institute for Advanced Study,
Princeton, New Jersey

Institute for Defense Analyses,
Communications Research
Division, *Princeton, New Jersey*

Intevep S. A., *Caracas, Venezuela*

Iowa State University,
Ames, Iowa

The Library of Congress,
Washington D.C.

Los Alamos National Laboratory,
University of California,
Los Alamos, New Mexico

Louisiana State University,
Baton Rouge, Louisiana

MacroSoft, *Warsaw, Poland*

Marquette University,
Department of Mathematics,
Statistics and Computer Science,
Milwaukee, Wiscon

Masaryk University,
Brno, Czechoslovakia

Mathematical Reviews,
American Mathematical Society,
Ann Arbor, Michigan

Max Planck Institut
für Mathematik,
Bonn, Federal Republic of Germany

McGill University,
Montréal, Québec, Canada

Michigan State University,
Mathematics Department,
East Lansing, Michigan

NASA Goddard
Space Flight Center,
Greenbelt, Maryland

National Institutes of Health,
Bethesda, Maryland

National Research Council
Canada, Computation Centre,
Ottawa, Ontario, Canada

Naval Postgraduate School,
Monterey, California

New York University,
Academic Computing Facility,
New York, New York

Nippon Telegraph &
Telephone Corporation,
Software Laboratories,
Tokyo, Japan

Northrop Corporation,
Palos Verdes, California

The Open University,
Academic Computing Services,
Milton Keynes, England

Pennsylvania State University,
Computation Center,
University Park, Pennsylvania

Personal TEX, Incorporated,
Mill Valley, California

Princeton University,
Princeton, New Jersey

Purdue University,
West Lafayette, Indiana

Queens College,
Flushing, New York

Rice University,
Department of Computer Science,
Houston, Texas

Roanoke College,
Salem, VA

Rogaland University,
Stavanger, Norway

Ruhr Universität Bochum,
Rechenzentrum,
Bochum, Federal Republic of Germany

Rutgers University, Hill Center,
Piscataway, New Jersey

St. Albans School,
Mount St. Alban, Washington, D.C.

Smithsonian Astrophysical Observatory, Computation Facility,
Cambridge, Massachusetts

Software Research Associates,
Tokyo, Japan

Space Telescope Science Institute,
Baltimore, Maryland

Springer-Verlag,
Heidelberg, Federal Republic of Germany

Springer-Verlag New York, Inc.,
New York, New York

Stanford Linear Accelerator Center (SLAC),
Stanford, California

Stanford University,
Computer Science Department,
Stanford, California

Talaris Systems, Inc.,
San Diego, California

Texas A & M University,
Department of Computer Science,
College Station, Texas

UNI-C, *Aarhus, Denmark*

United States Military Academy,
West Point, New York

University of Alabama,
Tuscaloosa, Alabama

University of British Columbia,
Computing Centre,
Vancouver, British Columbia, Canada

University of British Columbia,
Mathematics Department,
Vancouver, British Columbia, Canada

University of Calgary,
Calgary, Alberta, Canada

University of California, Berkeley,
Space Astrophysics Group,
Berkeley, California

University of California, Irvine,
Information & Computer Science,
Irvine, California

University of California,
Los Angeles, Computer Science Department Archives,
Los Angeles, California

University of Canterbury,
Christchurch, New Zealand

Universidade de Coimbra,
Coimbra, Portugal

University College,
Cork, Ireland

University of Crete,
Institute of Computer Science,
Heraklio, Crete, Greece

University of Delaware,
Newark, Delaware

University of Exeter,
Computer Unit,
Exeter, Devon, England

University of Glasgow,
Department of Computing Science,
Glasgow, Scotland

University of Groningen,
Groningen, The Netherlands

University of Heidelberg,
Computing Center Heidelberg,
Germany

University of Illinois at Chicago,
Computer Center,
Chicago, Illinois

University of Kansas,
Academic Computing Services,
Lawrence, Kansas

Universität Koblenz–Landau,
Koblenz, Federal Republic of Germany

University of Maryland,
Department of Computer Science,
College Park, Maryland

University of Maryland
at College Park,
Computer Science Center,
College Park, Maryland

University of Massachusetts,
Amherst, Massachusetts

University of Oslo,
Institute of Informatics,
Blindern, Oslo, Norway

University of Oslo,
Institute of Mathematics,
Blindern, Oslo, Norway

University of Salford,
Salford, England

University of Southern California,
Information Sciences Institute,
Marina del Rey, California

University of Stockholm,
Department of Mathematics,
Stockholm, Sweden

University of Texas at Austin,
Austin, Texas

University of Washington,
Department of Computer Science,
Seattle, Washington

University of Western Australia,
Regional Computing Centre,
Nedlands, Australia

Uppsala University,
Uppsala, Sweden

Villanova University,
Villanova, Pennsylvania

Vrije Universiteit,
Amsterdam, The Netherlands

Washington State University,
Pullman, Washington

Widener University,
Computing Services,
Chester, Pennsylvania

Worcester Polytechnic Institute,
Worcester, Massachusetts

Yale University,
Department of Computer Science,
New Haven, Connecticut

Individual Membership Application

Name _____

Institutional affiliation, if any _____

Position _____

Address (business or home (circle one)) _____

_____

City _____

State or Country _____ Zip _____

Daytime telephone _____ FAX _____

Email addresses (*please specify networks, as well*) _____

_____

I am also a member of the following other TEX organizations:

Specific applications or reasons for interest in TEX:

Hardware on which TEX is used:

Computer and operating system *Output device/printer*

_____

_____

_____

There are two types of TUG members: regular members, who pay annual dues of $60; and full-time student members, whose annual dues are $50. Students must include verification of student status with their applications.

Please indicate the type of membership for which you are applying:

☐ Regular @ $60 ☐ Full-time student @ $50

Amount enclosed for 1992 membership: $ _____

(*Prepayment in US dollars drawn on a US bank is required*)

☐ Check/money order payable to TEX Users Group enclosed

☐ Charge to MasterCard/VISA

Card # _____ Exp. date _____

Signature _____

Complete and return this form with payment to:

TEX Users Group
Membership Department
P. O. Box 594
Providence, RI 02901 USA

Telephone: (401) 751-7760
FAX: (401) 751-1071
Email: tug@Math.AMS.com

Membership is effective from January 1 to December 31 and includes subscriptions to *TUGboat, The Communications of the TEX Users Group* and the TUG newsletter, *TEX and TUG News*. Members who join after January 1 will receive all issues published that calendar year.

For more information ...

Whether or not you join TUG now, feel free to return this form to request more information. Be sure to include your name and address in the spaces provided to the right.

Check all items you wish to receive below:

- ☐ Institutional membership information
- ☐ Course and meeting information
- ☐ Advertising rates
- ☐ Products/publications catalogue
- ☐ Public domain software catalogue
- ☐ More information on TEX

AP-TEX Fonts

TEX-compatible Bit-Mapped Fonts Identical to Adobe PostScript Typefaces

If you are hungry for new TEX fonts, here is a feast guaranteed to satisfy the biggest appetite! The AP-TEX fonts serve you a banquet of gourmet delights: 438 fonts covering 18 sizes of 35 styles, at a total price of $200. The AP-TEX fonts consist of PK and TFM files which are exact TEX-compatible equivalents (including "hinted" pixels) to the popular PostScript name-brand fonts shown at the right. Since they are directly compatible with any standard TEX implementation (including kerning and ligatures), you don't have to be a TEX expert to install or use them.

When ordering, specify resolution of 300 dpi (for laser printers), 180 dpi (for 24-pin dot matrix printers), or 118 dpi (for previewers). Each set is on ten 360 KB 5-1/4" PC floppy disks. The $200 price applies to the first set you order; order additional sets at other resolutions for $60 each. A 30-page user's guide fully explains how to install and use the fonts. Sizes included are 5, 6, 7, 8, 9, 10, 11, 12, 14.4, 17.3, 20.7, and 24.9 points; headline styles (equivalent to Times Roman, Helvetica, and Palatino, all in bold) also include sizes 29.9, 35.8, 43.0, 51.6, 61.9, and 74.3 points.

The Kinch Computer Company
PUBLISHERS OF TURBOTEX
501 South Meadow Street
Ithaca, New York 14850
Telephone (607) 273-0222
FAX (607) 273-0484

Avant Garde Bold
Avant Garde Bold Oblique
Avant Garde Demibold
Avant Garde Demibold Oblique
Bookman Light
Bookman Light Italic
Bookman Demibold
Bookman Demibold Italic
Courier
Courier Oblique
Courier Bold
Courier Bold Oblique
Helvetica
Helvetica Oblique
Helvetica Bold
Helvetica Bold Oblique
Helvetica Narrow
Helvetica Narrow Oblique
Helvetica Narrow Bold
Helvetica Narrow Bold Oblique
Schoolbook New Century Roman
Schoolbook New Century Italic
Schoolbook New Century Bold
Schoolbook New Century Bold Italic
Palatino Roman
Palatino Italic
Palatino Bold
Palatino Bold Italic
Times Roman
Times Italic
Times Bold
Times Bold Italic
Zapf Chancery Medium Italic
Symbol ΔΦΓϑΛΠΘ
Zapf Dingbats ✂☞❑

TEX *without* Bitmaps

Wouldn't it be nice to be able to preview DVI files at any magnification, not just those for which bitmap fonts have been pre-built? Or to produce truly resolution-independent output that will run on any PostScript device, whether image setter or laser printer?

Perhaps you are looking for an alternative to Computer Modern? Well, there now exist complete outline font sets which include math fonts that are direct replacements for those in CM. Even if you do want to remain faithful to CM, there are distinct advantages to switching to the outline version of the fonts. We supply the tools to do all of this:

DVIWindo — *preview DVI files calling for outline fonts*

* Preview at arbitrary magnification
* Preview in Windows™ — a simple, standardized user interface
* Print to any printer with a Windows printer driver
* Show EPSF files with preview on screen — and insert TIFF images.

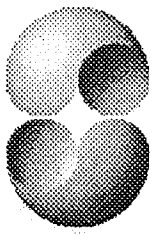
DVIPSONE — *partial font downloading for speed and efficiency*

* Avoid running out of memory on the printer
* Produce truly resolution-independent output
* Designed from the bottom up for use with outline fonts on the PC

Fonts — *available from Y&Y in Adobe Type 1™ form (ATM compatible)*

* BSR Computer Modern fonts — with accented characters built in
* LaTeX + SLiTeX fonts in outline form
* Euler font set — the most popular faces from the AMS font set.
* Lucida®Bright + LucidaBrightMath — a complete alternative to CM

Resolution-independent PostScript files using outline fonts can be printed by any service bureau, not just those with TEXpertise — and that translates into considerable savings for you. Perhaps it is time to get rid of those huge, complex directories full of bitmap fonts?



Y&Y, 106 Indian Hill, Carlisle, MA 01741 (800) 742-4059 (508) 371-3286 Fax: (508) 371-2004

Index of Advertisers

North America

ABRAHAMS, Paul
214 River Road, Deerfield, MA 01342; (413) 774-5500
Development of TEX macros and macro packages. Short courses in TEX. Editing assistance for authors of technical articles, particularly those for whom English is not their native language. My background includes programming, computer science, mathematics, and authorship of "TEX for the Impatient".

AMERICAN MATHEMATICAL SOCIETY
P. O. Box 6248, Providence, RI 02940; (401) 455-4060
Typesetting from DVI files on an Autologic APS Micro-5 or an Agfa Compugraphic 9600 (PostScript).
Times Roman and Computer Modern fonts.
Composition services for mathematical and technical books and journal production.

ANAGNOSTOPOULOS, Paul C.
433 Rutland Street, Carlisle, MA 01741; (508) 371-2316
Composition and typesetting of high-quality books and technical documents. Production using Computer Modern or any available PostScript fonts. Assistance with book design. I am a computer consultant with a Computer Science education.

ARBORTEXT, Inc.
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.

COWAN, Dr. Ray F.
141 Del Medio Ave. #134, Mountain View, CA 94040; (415) 949-4911
Ten Years of TEX and Related Software Consulting
Books, Documentation, Journals, and Newsletters
TEX & LATEX macropackages, graphics; PostScript language applications; device drivers; fonts; systems.

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.

ELECTRONIC TECHNICAL PUBLISHING SERVICES CO.
2906 Northeast Glisan Street, Portland, Oregon 97232-3295; (503) 234-5522; FAX: (503) 234-5604
Total concept services include editorial, design, illustration, project management, composition and prepress. Our years of experience with TEX and other electronic tools have brought us the expertise to work effectively with publishers, editors, and authors. ETP supports the efforts of the TEX Users Group and the world-wide TEX community in the advancement of superior technical communications.

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.

MAGUS, Kevin W. Thompson
P. O. Box 390965, Mountain View CA 94039-0965; (800) 848-8037; (415) 940-1109; magus@cup.portal.com
LATEX consulting from start to finish. Layout design and implementation, macro writing, training, phone support, and publishing. Can take LATEX files and return camera ready copy. Knowledgeable about long document preparation and mathematical formatting.

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 to meet publisher's or designer's specifications; instruction.

Outside North America

TYPOTEX LTD.
Electronical Publishing, Batthyány u. 14. Budapest, Hungary H-1015; (036) 11152 337
Editing and typesetting technical journals and books with TEX from manuscript to camera ready copy. Macro writing, font designing, TEX consulting and teaching.

July 27 to 30, 1992

13th Annual TEX Users Group Meeting

PORTLAND, OREGON

▲ Mark your calendars and join us in Portland, the home of 20-pound salmon and 20-story buildings. Ride light rail trains over cobblestone streets, ski Mt. Hood and attend the symphony in the same day—even in July. A friendly city, Portland charms its visitors with a variety of attractions including:

Windsurfing

A trip up the Columbia River on a sternwheeler

Tours of the wine region

The Metro Washington Park Zoo

Portland Center for the Performing Arts

Oaks Amusement Park

Oregon Art Institute

Scenic Washington County

Oregon Museum of Science and Industry

World Forestry Center

Mt. Hood

Portland Saturday Market for arts and crafts

Of special interest to TUG Meeting attendees may be the 11th Annual Mt. Hood Festival of Jazz to be held August 1st and 2nd in Gresham, Oregon, a suburb of Portland.

For a complete visitors' guide, *The Portland Book*, call the Portland Visitors' Center at (800) 345-3214.

TEX in Context

Resources, Support Tools, and Comparative Studies

▲ During four information-packed days, we'll delve into front-ends for TEX, inclusion of graphics within TEX documents as well as exportation of TEX output to other graphics programs, comparisons of implementations of TEX on microcomputers, network access and resources, educational issues, and translation between TEX and word-processors.

Presentations

Workshops

Networking Luncheons

Exhibits

Panel discussions

Classes

TEX USERS GROUP

▲ We'll meet and stay at the Benson Hotel, Portland's premier hotel recently restored to its grand stature of the early 1900s. A registered historic landmark, the Benson was built by Oregon lumberman, Simon Benson using elaborate craftsmanship and imported wood interiors. Special TUG rates: $89/night (available until June 26 only.)

▲ Program coordinator:

Mimi Lafrenz
ETP Services Co.

Program committee:

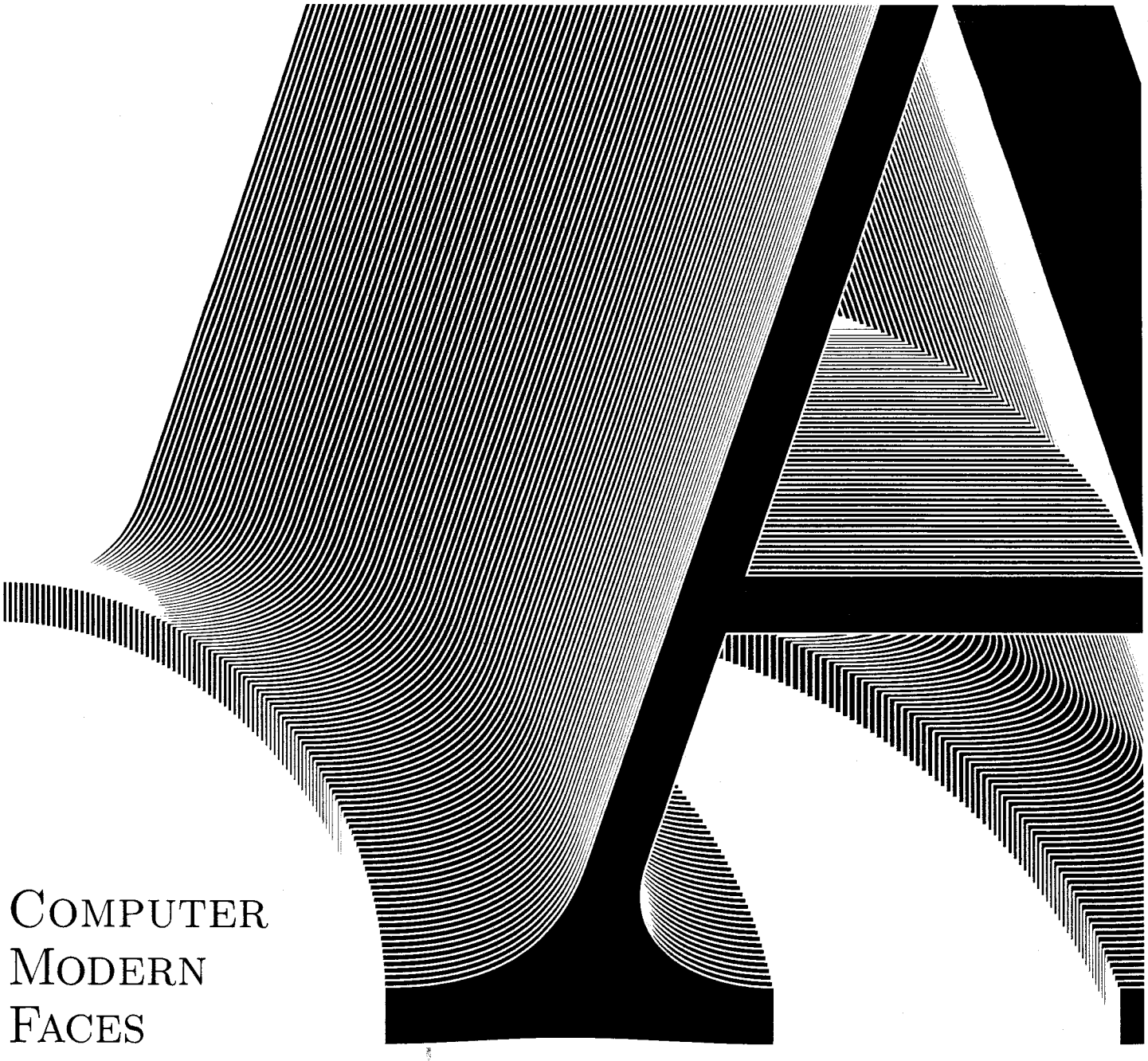
Helen Gibson
Wellcome Institute

Doug Henderson
Blue Sky Research

Ron Whitney
TEX Users Group

▲ Watch your mail and future issues of *TUGboat* and *TEX & TUG News* for more details. In the meantime, if you have questions, contact:

TEX Users Group
Phone: (401) 751-7760
Fax: (401) 751-1071
e-mail: tug@math.ams.com
P.O. Box 9506
Providence, RI 02940



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

TUGBOAT

Volume 13, Number 1 / April 1992