TUGboat

Volume 18, Number 4 / December 1997

TEX Users Group

Memberships and Subscriptions

TUGboat (ISSN 0896-3207) is published quarterly by the TEX Users Group, 1466 NW Front Avenue, Suite 3141, Portland, OR 97209-2820, U.S.A.

1998 dues for individual members are as follows:

- Ordinary members: $60; $55 if payment received before March 15, 1996.
- Students: $40; $35 if payment received before March 15, 1998.

Membership in the TEX Users Group is for the calendar year, and includes all issues of *TUGboat* 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 TUG elections. A membership form is provided on page 318.

TUGboat subscriptions are available to organizations and others wishing to receive *TUGboat* in a name other than that of an individual. Subscription rates: $70 a year, including air mail delivery.

Periodical-class postage paid at Portland, OR, and additional mailing offices. Postmaster: Send address changes to *TUGboat*, TEX Users Group, 1466 NW Front Avenue, Suite 3141, Portland, OR 97209-2820, 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.

Printed in U.S.A.

Board of Directors

Addresses
All correspondence, payments, parcels, etc.
TEX Users Group
1466 NW Front Avenue
Suite 3141
Portland, OR 97209-2820
USA

Telephone
+1 503 223-9994

Fax
+1 503 223-3960

Electronic Mail
(Internet)
General correspondence:
`TUG@mail.tug.org`
Submissions to *TUGboat*:
`TUGboat@mail.tug.org`

World Wide Web
`http://www.tug.org/`
`http://www.tug.org/TUGboat/`

In the bad old days, the index was a list of prohibited books; may we now, in a more enlightened age, ban books without indexes?

Stephen Jay Gould
An Urchin in the Storm (1987)

TUGBOAT

COMMUNICATIONS OF THE TEX USERS GROUP
EDITOR BARBARA BEETON

VOLUME 18, NUMBER 4 • DECEMBER 1997
PORTLAND • OREGON • U.S.A.

TUGboat

During 1998, the communications of the TEX Users Group will be published in four issues. One issue will contain the Proceedings of the 1998 TUG Annual Meeting. One issue, also not yet assigned, may be a theme issue.

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

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

Submitting Items for Publication

The next regular issue will be Vol. 19, No. 1. The deadline for technical items has already passed; reports and similar items are due by March 2. Mailing is scheduled for late March. Deadlines for other future issues are listed in the Calendar, page 315.

Manuscripts should be submitted to a member of the *TUGboat* Editorial Board. Articles of general interest, those not covered by any of the editorial departments listed, and all items submitted on magnetic media or as camera-ready copy should be addressed to the Editor, Barbara Beeton (see address on p. 235).

Contributions in electronic form are encouraged, via electronic mail, on diskette, or made available for the Editor to retrieve by anonymous FTP; 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 network FTP access, they will be sent on request; please specify which is preferred. Write or call the TUG office, or send e-mail to `TUGboat@mail.tug.org`.

This is also the preferred address for submitting contributions via electronic mail.

Reviewers

Additional reviewers are needed, to assist in checking new articles for completeness, accuracy, and presentation. Volunteers are invited to submit their names and interests for consideration; write to `TUGboat@mail.tug.org` or to the Editor, Barbara Beeton (see address on p. 235).

TUGboat Advertising and Mailing Lists

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

TUGboat Editorial Board

See page 235 for addresses.

Other TUG Publications

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

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

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.

MS/DOS is a trademark of MicroSoft Corporation
METAFONT is a trademark of Addison-Wesley Inc.
PC TEX is a registered trademark of Personal TEX, Inc.
PostScript is a trademark of Adobe Systems, Inc.
TEX and $\mathcal{AMS}$-TEX are trademarks of the American Mathematical Society.
Textures is a trademark of Blue Sky Research.
UNIX is a registered trademark of X/Open Co. Ltd.

Queries

Editor's note: When answering a query, please send a copy of your answer to the TUGboat editor as well as to the author of the query. Answers will be published in the next issue following their receipt.

– – ∗ – –

MusiXTEX: How many fonts are acceptable?

I have found problems with slurs in MusiXTEX when the altitude difference between slur begin and slur end is > 16 `\Internote` (i.e. more than one octave). But many scores need a larger slur extension. This can be done by splitting all slur fonts having 128 characters of the same convexity and slope into two fonts of 256 characters.

But the consequence is that the number of fonts will go from approximately 70 to 140 fonts, knowing that at the present time *all* fonts are invoked as TFMs when loading `musixtex.tex`.

So, here is my question to all TEXers (musicians or *not*):

> Do you know of a current implementation of TEX or any drivers which would be limited to 128 different fonts (instead of the usual value of 256)?

Here are the options:

- If everybody accepts 256 fonts of 255 characters, all is OK.
- If some people are limited to 128, I must maintain a "poor man's version" of MusiXTEX using *not more* than 128 fonts at a time.

Thanks in advance.

◇ Daniel Taupin
Physique des Solides
Univ. Paris-Sud
91405 Orsay, France
taupin@lps.u-psud.fr

Message from the President

Mimi Lafrenz Jett

Happy New Year!

Greetings, and best wishes for the upcoming year! It is hard to believe that this year has gone so quickly, and that we are already looking forward to the opportunities that 1998 will bring. It is going to be a great year for TEX users, with several important conferences planned throughout the year, and the release of *TEX-Live 3* scheduled for the first quarter of the year. The renewed energy of this group has sparked ideas and plans that are now coming to fruition, for the benefit of TUG members world-wide.

An outcry for more training, specifically for the everyday users of TEX and LATEX, was the basis for planning *TEX Northeast*, a conference in New York city to be held March 22–24. This is a new conference on our roster, not our annual meeting or a replacement for any other meeting, but a gathering based on need and the desire to work together. If you are going to be in the northeast USA in March, please plan to join us!

Speaking of great meetings, *EuroTEX 98* will be held in Saint-Malo, France, March 29–31. This annual event is hosted by GUTenburg, and will run successively with EP'98, the European electronic publishing event. Many in-depth seminars and talks will add to the experience as TEX users from all corners of the world come together for this exciting conference. Plan to attend *EuroTEX*, there will not be another meeting like it this year!

Our annual meeting and conference is planned for August 17–20 in Torun, Poland, hosted by the members of GUST. This will be our main conference for the year, with presentations from TUG members covering all aspects of our technology, and gatherings designed to improve communication and understanding between our people. This is one of the important benefits of our meetings, having the chance to know each other and share the work we are doing. If you are able to attend any of these events, please plan to join us. It will be an experience you will not quickly forget.

With the new year, we have a chance to heal the wounds of the past and move forward. There have been many mistakes, and misunderstandings in our past, as in any organization. It is time to put them aside and focus on our objectives. In the time I have worked with the board, beginning in 1992 and through the present day, I have witnessed intense emotions and some hurt feelings. But I have also seen a spirit of cooperation and goodwill that extends from and to every person involved with TEX. It is good to note that the positive outweighs the negative, and is increasing at a steady rate. My sense for 1998 is that we are growing closer to the type of group we have hoped to become, and that we can put the past into the history books and enjoy the future as it unfolds. We expect increased membership this coming year, and growth in many areas to support that membership.

The business affairs of TUG are under scrutiny, as we try to bring closure to the accounting questions that have bogged us down in the past year. At this time, we are undertaking an internal audit of our records to ensure accurate reporting to you and our friends in the government(s). It appears that a bug in our accounting software has left many accounts out of balance and confused the outcome. The 1996 tax return was filed, at the advice of our CPA, but we know that there are errors. As a non-profit organization, the penalties are steep for non-filing so we prepared the return with the understanding that we will amend it as soon as we know the exact numbers. A report of our financial condition will be posted when it is complete. The good news is that we are enjoying positive cash-flow due to our diminished expenses, and the membership renewals are already coming in daily from individuals and institutions. We finally have a system in place for tracking memberships and accounting on an individual basis.

On a personal note, the work and employees of ETP/Harrison have become a part of Interactive Composition Corporation, another TUG member for many years. In fact, I originally met Gordon Johnson at the TUG meeting in Dedham, Mass., in 1991. He was a friendly competitor, always willing to work to improve the conditions for all of us. After many years of struggling in the publishing industry, it was a welcome relief to entertain the thought of selling to Gordon and his company. We are looking forward to a long and wonderful relationship. Concerning the TUG office, ICC has generously agreed to continue to house the office at no charge to the organization (other than telephones and staff) for the duration of our lease on Naito Parkway. Thanks, Gordon!

Here's to you, and the new year, may it be full of hope and inspiration. Thank you for your support and good wishes this past year.

◇ Mimi Lafrenz Jett
ETP Harrison, 1466 NW Front Avenue, Suite 200
Portland OR 97209-2820 USA
mimi@teleport.com

General Delivery

Editorial Comments

Barbara Beeton

New TEX groups: TUGIndia and TUG-Philippines

On November 16, 1997, the Indian TEX Users Group was informally launched at the academic premises of the Department of Mathematics of the University of Kerala, Trivandrum. The session was chaired by Prof. K.S.S. Nambooripad, "a world renowned mathematician and an ace TEX programmer"; he was unanimously elected the first chairman of the new group. The other officers are C.V. Radhakrishnan, Secretary, Dr. R. Rajendar, Treasurer, with an executive board of nine other academics; it was planned to invite three representatives of the TEX-using publishing industry to round out the executive committee.

The formal inauguration ceremony has been set for January 1998, with Sebastian Rahtz, representing UKTUG and Elsevier Science, agreeing to be present for the occasion. The group plans to circulate a quarterly newsletter, TUGIndia, and to sponsor courses and seminars. Information can be obtained by sending e-mail to `tugindia@mailexcite.com`.

A report of the inaugural meeting will appear in the March 1998 issue.

The other "new" group has actually been in existence since 1993, although we have only recently learned of its existence. TUG-Philippines will sponsor its third nationwide seminar-workshop in May 1998. The president, Dr. Severino V. Gervacio (`cossvg@mail.dlsu.edu.ph`) reports that they are in the process of creating a Web page with information about the group.

TUG CTAN host and Web site

The addresses for the TUG host machines have finally stabilized. The CTAN host is on one machine, and can be reached either by FTP or a Web browser at

`ctan.tug.org`

The Web site is on a second machine, at the URL

`http://www.tug.org/`

The Web site contains links to many other sites, including those of all the local user groups, TEX suppliers (CTAN, other sources of free and shareware, and commercial vendors), and is updated regularly as new information becomes available. Your comments and suggestions for additions to the site are welcome.

SGML extensions — update

Draft proposals for the Mathematical Markup Language (MathML) and the Extensible Markup Language (XML) have been submitted to the World Wide Web Consortium (W3C) for balloting. XML was granted Proposed Recommendation status on December 8, 1997, and a similar status is expected for MathML early in 1998. (These two tools were described briefly in last issue's *Editorial Comments* column.)

John Bosak, chair of the XML Working Group, made a presentation at the San Francisco TUG meeting, but was unable to complete a paper for the proceedings. During most of 1997, the technical landscape was changing so rapidly that a static version of Jon's presentation would not be accurate, and he preferred to wait and revise the material in light of the specification as finally approved. Since the draft of XML had reached Proposed Recommendation status and was undergoing balloting at the end of the year, this goal is in view. An update should be available for the March issue of *TUGboat*.

One significant honor can be announced, however. The XML Working Group was named a recipient of the 1997 Seybold Editors Award. The 19 winners "include products, technologies and companies that have already made a notable impact, or are likely to have a substantial effect, on the future of online and print publishing." The portion of the announcement relating to XML reads:

> W3C's XML Working Group.
> A working group within the W3C created a simplified form of SGML called the Extensible Markup Language, or XML. With the initial XML specification complete and vendors lining up to support it, we salute the XML Working Group for creating a metalanguage that will improve text processing, searching and composition on the Web.

Readers seeking more information on MathML and XML can consult the following URLs:

`http://www.w3.org/TR/WD-math/`
`http://www.w3.org/XML/`
`http://www.ucc.ie/xml/`

◇ Barbara Beeton
American Mathematical Society
P. O. Box 6248
Providence, RI 02940 USA
`bnb@ams.org`

Dreamboat

ε-TEX V2: a peek into the future

Philip Taylor

Abstract

ε-TEX V1 was released towards the end of 1996, and it was intended at the time that ε-TEX V2 should be released approximately one year later. For various reasons the release date has slipped a little, but V2 is in alpha-test and we confidently expect to release it in the near future: indeed, it may well have been released by the time that this article appears in print. In this article the new features of V2 are reviewed, and we conclude by peeking a little further into the future to see what will happen with $\mathcal{NTS}$.

Introduction

When the $\mathcal{NTS}$ project was first established in 1992, we hoped that work would commence fairly quickly. Sadly it became all too obvious that a project of that magnitude would require one or more full-time workers, and none of the (volunteer) team had that sort of time to spare. The team decided that, until funding could be found, they would work on the less-demanding but still worthwhile task of extending TEX in its current (Pascal-web) form. The first fruits of that work were revealed in late 1996, when ε-TEX V1 was released. This version added approximately 30 new primitives to TEX and a small adjunct macro library was also produced which both extended the plain TEX format to accommodate the new primitives and also added new functionality such as natural language handling and TEX module libraries.

Once ε-TEX V1 had been released, the group were free to concentrate on the next version. Originally intended to ship a year later than the first, this version has slipped a little as pressure of other commitments has forced members of the team to invest less time in the project than they would have wished. However, V2 is now in alpha test, and we confidently expect to be able to make a general release in the near future: release may well have taken place by the time this article appears in print. The majority of the remainder of this article describes the features that we are fairly confident will be present in that release.

Ideas which are almost certain to appear in ε-TEX V2. Although we do not wish to give an absolute commitment at this stage, particularly as many of the proposals are the process of being tested at the time of writing, we do believe that the ideas contained in the following section are very likely to appear in ε-TEX V2. A subsequent section discusses ideas which may appear in future releases.

Increasing TEX's registers. TEX has 256 of the most commonly used registers: counts, dimens, skips, boxes, toks, etc., and whilst these are enough for normal applications, advanced formatting systems really require more. In ε-TEX V2, we intend to provide 32768 of each of these, which we hope will be sufficient for the most demanding packages. Insertion classes will still be restricted to 256 or fewer, and `\box` 255 will retain its special significance. The "etex" format will allow both local and global allocation of these registers ("plain" allows only global), and for efficiency reasons a user will be able to elect whether to allocate a register from the dense (0..255) pool or from the sparse (256..32767). To allow the allocation mechanism to overflow from dense to sparse without risking a conflict with the allocation of insertion classes, the format allows a user or package to pre-reserve a number of insertion classes. Facilities for block-allocating a contiguous set of registers will be provided.

Improved natural language handling. TEX overloads the `\lccode` concept, using it both for "real" lower-casing operations and also for purposes of hyphenation. In ε-TEX V2 these operations are unbundled, and the codes used for hyphenation can be staticised as the patterns are read in (the current set of lccodes is used). Thereafter, whenever a particular language is used, the corresponding set of hyphenation codes is loaded.

Arithmetic expressions. Although TEX can perform simple arithmetic (addition, multiplication and division), these operations are in general "assignments" and therefore cannot be used in expansion-only and certain other contexts. ε-TEX V2 provides a set of arithmetic primitives which evaluate an expression in such a way that the value of the expression can be accessed in expansion-only contexts, as well as being usable (for example) when TEX is looking for a ⟨*number*⟩, ⟨*dimen*⟩, etc. As TEX intentionally uses only integer arithmetic wherever the results of a computation are accessible to the user, floating-point arithmetic has *not* been provided. There are four new primitives, `\numexpr`,

\dimexpr, \glueexpr and \muexpr, each of which requires its operands to be of appropriate type (or coercible to that type). Parentheses may be used to indicate precedence wherever this will clarify or disambiguate an expression. The normal arithmetic operators "+", "-", "*" and "/" are allowed within an expression.

Discards are no longer discarded!. When TEX performs page-breaking, so-called "discardable items" which follow the chosen breakpoint are discarded; whilst this is perfectly reasonable if the page break is actually taken, recursive techniques aimed at optimising the appearance of multiple pages require the ability to "undo" a pagebreak in order to try the effect of breaking elsewhere. The discarded items are therefore required in order to re-create the vertical list which TEX is trying to break. In ε-TEX V2, we allow access to these "discarded items" via a new primitive \pagediscards. The discards which occur during \vsplitting are also accessible via an analogous primitive \splitdiscards. Both of these primitives return a vertical list, similar to that obtained by \unvboxing a box register.

Read-write access to \parshape. Although TEX allows the user to create arbitrarily complicated paragraph shapes through the use of the \parshape primitive, it provides no way for the user to find out which \parshape is currently active (although it does allow the user to ascertain the number of lines of the current \parshape specification). In ε-TEX V2, we allow full read access to all the elements of the current \parshape.

Interrogating the current conditional context. In ε-TEX V1 we allowed users to make environmental enquiries concerning the current group, both as to its depth of nesting and to its type. In ε-TEX V2, we generalise this concept and allow analogous access to the current conditional context through the use of \currentiflevel, \currentiftype, \currentifbranch and \showifs.

Access to information concerning font-character combinations. Although it is possible to gain some information about a particular character in a given font by typesetting that character in a box and then measuring the dimensions of the box, not all the dimensions of the character can be reliably obtained in this way, and there is no way to ensure that the character actually exists in the font before attempting to typeset and measure it. In ε-TEX V2 we allow the user both to check whether a particular character exists in a given font, using \iffontchar, and (if it does exit) to measure the four fundamental dimensions of that font/character combination using \fontcharwd, \fontcharht, \fontchardp, and \fontcharic (representing width, height, depth and italic correction respectively). Furthermore, we ensure that users are alerted to the existence of missing characters in a font by causing lost characters to be logged to the console as well as to the log file if \tracinglostchars is set to a value greater than 1.

Better debugging aids. In order to assist in diagnosing mis-matched or runaway group problems, ε-TEX V2 allows the user to opt to be warned whenever a file is left in a group or conditional other than that at which it was entered. This may be accomplished by setting \tracingnesting to a value greater than zero.

Subtle change to the semantics of \protected. ε-TEX V1 introduced a new prefix, \protected, which inhibited the expansion of the "protected" macro in contexts in which expansion was unlikely to be required. Further research into this area suggested that at least one such case had been missed, and "protected" macros are now inhibited from expansion when TEX is scanning ahead while processing alignments.

Optimisations. To improve the overall efficiency of ε-TEX internal modifications have been made to reduce the resources required when there are a number of \aftergroups active for a single group, and to eliminate the stack space wasted in setting a register to the same value as it currently holds.

Access to the components of a glue quantity. Whilst it is possible to gain access to the various components of a glue value by clever macro programming, the code required is sufficiently arcane to suggest that a better method is much to be preferred. Accordingly we are considering a set of primitives \gluestretch, \gluestretchorder, \glueshrink and \glueshrinkorder which will give much-simplified access to these quantities. As a part of the same process we are looking at two conversion primitives, \mutoglue and \gluetomu.

Improved typographic quality. Whilst the majority of the work in ε-TEX is aimed at providing the ε-TEX programmer with more powerful tools, we are aware that the real purpose of TEX is to generate typeset output of the highest quality. During a meeting in Brno with Prof. Knuth on the occasion of his honorary doctorate, he suggested that we

might like to consider improving the typographic quality of the last line of a paragraph. According to Don, traditional (hot-lead) typesetters would set the last line to the same tightness or looseness as the immediately preceding line, and he thought that ε-TEX should be capable of doing likewise. We are looking into providing this but in a parameterised manner, so that *all* possibilities between TEX's current behaviour and that suggested by Don can be achieved. We think that this might be controlled by a parameter called `\lastlinefit`.

Improved typographic quality, cont.. In the same vein, we are looking into ways of allowing better parameterisation of the page-breaking process by having not just one penalty for (say) a club- or widow line but a whole array of such penalties which can reflect the undesirability of leaving one, two, three to n lines at the top or bottom of a page. Other related penalties are also candidates for this process.

Ideas still under discussion

The following ideas are all under discussion but are very unlikely to find their way into ε-TEX V2: some may be deferred to ε-TEX V3, and some may never appear at all. Although the group have some idea into which category each of these ideas may fall, it is probably not helpful to go into too much detail here, and so they are all lumped together as "under discussion".

Can TEX find this font?. In "the good old days", a TEX program could count on finding all 76 of the standard TEX fonts no matter where it was run in the world. These days, with many documents being set in exotic fonts from a myriad of sources, it is no longer certain that, just because site A has font F, site B will have the same font. We are therefore considering providing an `\iffont` primitive which will allow ε-TEX to ascertain at run-time whether a particular font exists on the system on which the document is being processed. It is not certain at this stage whether this would be a simple "does this font exist?" test, or a more complex "does it exist and is the TFM file for it valid?". `\tryfont` has also been suggested as an alternative approach.

Maths alignments. Peter Breitenlohner, the implementer of ε-TEX, probably typesets more mathematics than the rest of the group put together, and he believes that there is a case for a maths alignment primitive. He has not yet finished his research on this topic, and all that can be said at this stage is that we are considering implementing some form of `\malign`.

Typesetting on a grid. Whilst TEX is *excellent* at typesetting in designs where variable quantities of white space can be allowed to occur, trying to coerce it to set on a regular grid (something at which packages such as Quark Xpress excel) is *far* more difficult. The various macro-based solutions which have been tried do not seem to address the underlying problems, and we are looking at providing an entirely new paradigm within ε-TEX whereby material being typeset can be caused to "lock on" to a grid at some point in the page-building process. Although at first sight it might be thought that it is the reference points of the lines making up the page which need to lock to this grid, we are fairly certain that this is not always the case, and we are therefore looking at ways of associating one or more "handles" with a particular box. In the degenerate case there will be one handle which is coincident with the reference point of the line, but in more complex cases there may be two, three or ever more handles, each of which will lock on to one line of the raster. Even so, there are also situations in grid-based designs where the grid-lock contraints just have to be violated, and one topic still unresolved is whether it is then better simply to allow the box to float free, or whether it is better to constrain it in some way, perhaps by associating with the handle(s) a degree of flexibility which is in some way analogous to TEX's current use of the "glue" concept.

More on improved typographic quality. Another point made by Don during his stay in Brno is that there are situations in which TEX's (vertical) positioning of elements of mathematical formulæ is less than ideal. He points out that even in typesetting the TEXbook he had to make use of kludges such as `\sub \strut` in order to achieve the best visual effect. We are investigating ways in which the effect (and related effects) could be achieved by better parameterisation of the mathematical typesetting process.

NTS

In the introduction to this paper it was mentioned that the $\mathcal{NTS}$ project proper had been put "on ice" until the group had sufficient funds to allow a programmer to be employed full-time to work on the project. It is with great pleasure that I can now report that, as a result of the generosity of DANTEeV the group has DM 30 000 which can be

used for this purpose. On the recommendation of Jiří Zlatuška, we have made an offer to Karel Skoupy of the Czech Republic, which he has accepted, and Karel will be starting work on $\mathcal{NTS}$ during late February 1998. It has been agreed that the language of implementation will be Sun's JAVA, and Karel's first task (apart from becoming a JAVA expert...) will be to draw up a specification for $\mathcal{NTS}$. Provided that the group agree with his design, he will then start work on implementing $\mathcal{NTS}$, and we hope to be able to review his work after a further six months. Within one year of commencement we hope to have a working implementation of $\mathcal{NTS}$ not simply a port of TEX to JAVA but a total re-design intended to emphasise the deep structure of TEX whilst avoiding the design features which make the present system rather difficult to extend or change.

The group are still determined that $\mathcal{NTS}$ will be 100% TEX-compatible, and are confident that it will remain so for at least the first five years of its life. We are less certain whether divergence should then be permitted, in order to add new functionality which is in some way incompatible with TEX. If we do decide that compatibility must be sacrificed, we will give considerable notice of that decision, and users who must retain the ability to process legacy documents in a manner identical to TEX will be advised to take an archive copy of $\mathcal{NTS}$ before compatibility is lost.

One exciting idea which the use of JAVA permits is the possibility to integrate access to CTAN (CNAN?!) with $\mathcal{NTS}$: it is by no means impossible that $\mathcal{NTS}$ might be able to fetch for itself any module which cannot be found on the local system and which is needed in order to process a document. If that becomes a reality, TEX will have become truly integrated into today's (and tomorrow's) globally-networked world.

◇ Philip Taylor
Royal Holloway and Bedford New College, University of London
Post: Room 106, Computer Centre,
Egham Hill, Egham
Surrey TW20 0EX, United Kingdom
P.Taylor@Vms.Rhbnc.Ac.Uk

Typography

Typographers' Inn

Peter Flynn

1 Return to the fold

I'm delighted that the plan to incorporate parts of the former *TEX and TUG News* has now come to fruition, and credit is due to the editors for their hard work on this. I know that when it was first suggested there were objections from several people that it would in some way dilute the TEXnical excellence of *TUGboat* to include articles about anything except internal TEXnicalities, but I am happy to see from recent issues that this has not been the case.

On the contrary, I think *TUGboat* has benefited from the cross-fertilization that the change has involved. Given that computer users have easy access to other (lesser!) typesetting systems, I believe it is important that users, companies, libraries, universities, and the printing and publishing industries see that the TEX community is continuing to research and publish better ways of doing the job, in *all* fields, not just the mathematical and scientific.

2 H&J

No, it's not a fetish like S&M (or even a genre like S&F), but the compositor's abbreviation for hyphenation and justification. Those who have had to produce publication-quality typesetting will be aware of the problems raised by setting in short measures like narrow newspaper or magazine columns, especially when you have to include lengthy names from a computing environment which include a 'path' (the hierarchy of folder names which provides the route by which you can locate a file), for example: `Hard Disk:Clients:Acme Cookies Inc:Projects:Bakery:Progress Report January 1998`.

While names with embedded spaces can be broken at a space, it may not always be clear to the reader that there was originally a space at what has become a line-break. Many publications still seem to use typesetting systems which fail grotesquely to break at sensible places, especially in names with no embedded spaces, such as URLs. Achieving a smooth and even finish is admittedly difficult in texts with many reluctant hyphenations, but the reason I bring this up again is in answer to those of you who have written to me asking for ammunition

to use against well-meaning but ignorant publishers who insist on inserting hyphens or breaking URLs *before* punctuation (putting periods *etc* at the beginning of lines!). This looks terrible but most of them seem oblivious of it. If you want to hit them with chapter and verse, therefore, it's the path algorithm and macro by Nelson Beebe and Phil Taylor that I have referred to before. It works like `\verb`, it's in eplain and there's a path package for LaTeX as well, and it also forms part of the url package used for *TUGboat*. It makes *un*hyphenated breaks, and path allows you to specify the characters (usually punctuation) after which breaking is to be allowed, for example:

```
\discretionaries +\:/.$%&;=@_?#|~+
```

However, all such sets of characters have some drawbacks, and there are a few hidden traps for the unwary. The rules for url are documented in the style file and were chosen for the *TUGboat* production environment.

- For path, you may want to restrict the set of characters for each document, as there may be some that you want to exclude from being breakpoints. I use the example above for some of my own work, but your mileage will vary: plus, exclamation mark, and double-quote do occur in email addresses, and you may want angle brackets or curly braces if you need to illustrate markup.
- Bear in mind that folder path-names should never, never hyphenate at a genuine embedded hyphen, such as in `emacs-19.27.tar.gz`, because of the risk of confusing the reader, who may not then be able to work out if the hyphen belongs to the name or if it has been inserted by the typesetting process.
- None of the implementations appears to let you distinguish between *a*) characters which must be treated *verbatim* simply because they are literals which you do not want TeX to interpret on some occasion, and *b*) those characters (perhaps including some of those required *verbatim*) which are being specified as potential breakpoints. There thus appears to be no way to say 'break at a period or a comma or a slash but not at a backslash or percent sign or hash mark (pound sign); but still allow me to include backslashes and percent signs and hash marks as verbatim characters.' If you want to use `\path` for URLs, for example, you must include the tilde, because it occurs in some URLs as a 'personal directory' flag character for usernames, even though it would be a semantic error to break a URL after such a tilde. Personally I'd also rather not have breaks occurring after the question mark, hash sign, or vertical bar (XML) separating a URL path from its query or locator string, but if you don't include them, the macros will gag when you feed it a URL containing one.
- An unfortunate by-product is that the start of a URL falling close to the end of a line may break somewhere within the hyphen and the two slashes, whereas the ideal place would be after the second slash; or that TeX control sequences may break *after* the backslash!

It would be nice if some byte-level guru could beef up the selectivity of the algorithm to let it cater for string exceptions, and also add some kind of interpretive shield which will let it gobble *all* characters *verbatim* except the sentinel (the character enclosing the definition), but never break after them (but this would then mean no documenting both TeX control sequences and PC folders with the same setting...)

3 Quote unquote

Has anyone any idea where the English use of the curious mirror-quote comes from? English quotation marks are 66s and 99s, as all TeX users learn, but I see more and more books using open-quotes which are lateral mirrors of the 99s, **cc**s as it were, ‘like this’. I have hundreds of fonts here, but I can't find any which contain it. It's a perfectly normal quote in some languages, and it has its own Unicode (ISO 10646) code point (`u+201d`, 'single high-reversed-9 quotation mark' or 'single reversed-comma quotation mark') but how it migrated into use in English is a mystery, but there is one of them in the `wsuipa` fonts at `\char'163`.

I first noticed it a long time ago, in *'Rommel?'* — *'Gunner Who?'*, the second volume of Spike Milligan's WWII autobiography, published in London in 1974 by Michael Joseph and printed by Hollen Street Press in Slough in what looks like Century Schoolbook. These open-quotes are used throughout, except for the 'Rommel?' on the cover and title page, which is in Fraktur, and has the open-quotes the right way round (as do the sans-serif sidebars). It may be too long ago now to find out, but I've seen it dozens of times since, in books and other documents. Typographically it's a nonsense in English, and if this was still the days of filmsetting I'd say someone put the mat in back to front, which ought to be impossible, but as it has persisted into DTP I'd be interested to know who's doing it and why.

4 Encoding variations

Some time ago I bought the MathTime fonts from Y&Y, to do Times math setting. It was a bit of a struggle to install them, as I use UNIX most of the time, and their installation assumes *a*) a PC, and *b*) Adobe Type Manager. I finally did it, though, and I'm going to document it for posterity soon, but it raised some additional questions about the use of PostScript font encodings in TEX.

I use a lot of PostScript fonts from sources such as the Monotype and other CD-ROMs. Most of my work has been in English, using plain TEX, so I've been content with my homebrew `psfonts.tex` macros to handle the occasional accent, and the rare time or two I needed LATEX, I stuck with the OT1 ('old TEX') default encoding. (For those of you not using PostScript fonts, 're-encoding' is needed to reconcile the fact that Adobe uses a different font layout from the TEX one.) Having to load the Y&Y fonts for a LATEX job, however, meant choosing a more comprehensive solution. Their preference is to use their own 'TEX'n'ANSI' encoding, and as I had no personal axe to grind, I picked this in order to minimize any installation problems.

I've also been re-encoding a few other PostScript fonts. I'm happy to report that it's quite painless, and I've only hit one or two small kinks (my æ vanished for some reason). So much so that I've revised my script which copies a font off a CD-ROM, passes a dead chicken over the keyboard, and mutters the incantations about `afm2tfm` and `vptovf` necessary to make it TEXable. This is now at `ftp://ftp.ucc.ie/pub/tex/mkvf` (with a PC version, `cdvf.bat`, in the same directory). If you haven't started using PostScript fonts, it's well worth it, especially as the CM fonts are now available in that format, saving you acres of disk space in bitmaps. Now that I think I've grasped the way you specify additional fonts with PSNFSS, it has freed me from the use of CM in LATEX (and there was much rejoicing!).

5 TEX and TEXability

I have long resisted using LATEX simply because of the pain of having to undo almost everything it does in order to make it work the way that real-world publishing needs. Plain TEX with Karl Berry's eplain provides many of the necessary textual features (cross-referencing, indexing, citation, *etc*) in a more usable manner without the heavily idiosyncratic default layout styles of LATEX. However, the range of packages now available for LATEX 2ε, and my re-reading the LATEX 2ε version of Lamport's *LATEX: A Documentation Preparation System* as well as *The LATEX Companion* over the summer, persuaded me to try one more time to come to terms with it, and it has not been unsuccessful.

The *Companion* is crucial: it's the only reliable source of explanatory information in one volume about all the major packages as well as much of the guts of LATEX 2ε. A copy of the new edition of Lamport's book is also required, as it covers some of the extremely basic things I didn't know, some of which are not covered in the *Companion*. In the past I have found that even for relatively short jobs like articles for a journal, you can need a surprisingly large number of packages, and for a book you may need a dozen or more. What did please me was that most of the packages I have used so far are pretty much bug-free, well-documented, and perform as advertised, even in combination. Well done, the authors.

The stumbling-blocks are a mixed bunch. LATEX 2ε has roughly the same default behaviour and appearance as LATEX 2.09, so anything even mildly different needs fairly verbose recoding. I only came across four serious annoyances (and I've no doubt that I will be severely treated for airing them if I've missed something):

- As any reader of `comp.text.tex` knows, it's the simplest changes which are the most difficult to find out about for the beginner: try looking up how to shift the left margin inwards by the value of `\parindent` for the duration of a verbatim environment. It's probably simple, but it doesn't seem to be obvious. Hanging footnote indentation is another quite common requirement: it *is* in the *Companion*, but as with Lamport's book you have to hunt for it (p.73, called a 'complicated variant'!).
- Documentation of float control is much improved in LATEX 2ε, but it's still relatively opaque: it's sometimes hard to understand *why* your figure has wandered off by three pages (`\begin{tracing}[floats]`...`\end{tracing}`?). One problem is still that a figure has to fit on a single page, max. Long examples of source code, even set `\tiny`, can run to several pages, but for some reason LATEX does not seem able to break them onto multiple pages.
- Contextually-sensitive hyperlinks are not a luxury. When I'm writing, I really don't want to bother having to remember if the phrase I want to refer to is in a chapter, or a section, or a subsection, or a figure, or a table, or a sidebar, or an example, or whatever. I know the label I gave it, and I expect a cross-reference system to know the type of object for me, and

to insert the relevant phrase or symbol accordingly; and if I move the target from a figure to an example, I expect the point of reference to reflect this. In other words, `\ref{foo}` must produce 'Figure 7.4' when `\label{foo}` occurs inside a figure, and 'section 3.5' or 'chapter 9' or whatever when it is located in a section or a chapter. In a sidebar, I would expect it to retrieve the title and page number: 'see the box "Cattle rustling" on page 173'. In complex technical or analytic texts this can save days of work: the facility is built into eplain. LaTeX already knows the environment for any `\label`, so it should be possible.

- Lastly, there must still be a way to get better parameterisation of the typographic dimensions that standard compositors' specifications require. Cutting and pasting large chunks of code (whole macros) just to change a value from 3pt to 4pt is not the way to go. The control over formatting changes such as the appearance of sectional headings is already excellent, and many packages extend this level of control, but more are needed.

The work done by the *Companion* is magnificent, but there is still a tendency for some other documentation to concentrate on the 'borderline cases and special parameters' rather than the daily necessities: as I said earlier, some of the most elementary requirements are still entirely non-obvious, even to someone familiar with TeX. Anyway, I'm almost a convert. I'm going to use LaTeX 2_ε for my forthcoming book on SGML (plug, plug) and we'll see how it copes.

6 Humanities citations

The one big thing I did have to hack together just before Yuletide was a new citation style. I was setting a thesis for a colleague in the Humanities, and this is still a neglected area for TeX and LaTeX. The edmac system does a great job for the annotation of *varia* and the *apparatus criticus*, but there are still just as many citation styles as in the natural sciences. One of the most common is the footnote, and of the two packages available, camel and footbib, the first doesn't yet do what I needed, and the second tries to do too much and requires the use of additional control sequences: I wanted something transparent. The style is so prevalent that historians actually use the term 'footnote' to mean 'citation' (I haven't yet found out what term they use when they want to talk about 'normal' footnotes.[1]) In this style, you would refer for example to the `\nutshape` control sequence,[2] as well as giving the full citation in the list of bibliographic references at the end.

The requirement, therefore, was to let the author type `\cite{foo}` in the normal way, and have the bibliographic style retrieve the short citation in author-title-year format to be put into a standard footnote with superior numbering, rather than put author and year or a reference number in brackets. But that wasn't all: if the immediately following citation is to the same work by the same author, then all the footnote should say is [3]. And if the same work was cited again later in the document,[4] the footnote should just say [5].

In both cases you also need to be able to add an optional comment or page number, but this is adequately handled by `\cite[options]{foo}` (although putting square brackets themselves in the optional text caused some interesting error messages). I used the chicago style, renamed it human and hacked the `.bst` file to output the new citation instead of the traditional author and year. This last bit was the worst: the documentation on BibTeX styles is thin, most styles themselves are uncommented, and the BibTeX language itself, while ingenious and efficient, is obtuse and poorly documented. However, the fix worked, to the accompaniment of reams of BibTeX error messages about the stack, which I don't grok,[6] but which don't seem to affect the result. Once I've hammered out the error messages I'll make it available and perhaps some reverse Polish gureaux can find my errors in time for TUG's Polish sojourn.

◇ Peter Flynn
Computer Centre,
University College Cork,
Ireland
pflynn@imbolc.ucc.ie
URL: http://imbolc.ucc.ie/~pflynn/

[1] Like this one

[2] Goossens M, Mittelbach F, and Samarin A. *The LaTeX Companion*, 1994, p.56

[3] *ibid.*

[4] But with an intervening reference or footnote about something else.

[5] Goossens M *et al*, 1994, *op. cit.*

[6] I mean I know about stacks, but I can't work out why what I've done should affect it.

Book Review

Book review: *The LaTeX Graphics Companion*

David F. Rogers

Michel Goossens, Sebastian Rahtz and Frank Mittelbach, *The LaTeX Graphics Companion*. Addison Wesley Longman, Reading, Massachusetts, 1997, ISBN 0-201-85469-4.

Before discussing the book it is best to establish the reviewer's prejudices. I normally use Plain TeX rather than LaTeX. However, I do a bit of computer graphics so when Barbara Beeton asked me to review *The LaTeX Graphics Companion* I agreed.

The book begins with brief discussions of the principal graphics packages available for LaTeX. These brief discussions are followed by quite thorough discussions of these graphics packages:

- the LaTeX 2_ε graphics bundle,
- METAFONT and METAPOST,
- PSTricks,
- Xy-pic,
- XYMTeX,
- ppchtex,
- FeynMF,
- MusiXTeX

A brief discussion of the chess font and `chess` package is also included.

The use of color in LaTeX 2_ε is clearly discussed. Fonts are always a bit of a problem in (La)TeX. There is a useful discussion of PostScript fonts and the PSNFSS system along with a discussion of font manipulation tools and installation issues. The final chapter discusses PostScript drivers including `dvips`, `psutils` and Ghostscript.

As an example of the level of detail in many of the discussions of the various packages, the PSTricks chapter includes discussions of the basic components and concepts, the commands and how they are structured, the coordinate system used and the use of color. Numerous examples of graphics and mixed graphics and text, nodes and trees, basic data plotting and basic three dimensional graphics are given along with the LaTeX source that created them. A short discussion of customizing PSTricks is included. The chapter concludes with a useful list of PSTricks commands.

This is a useful book for anyone who uses any variant of TeX because the concepts presented are applicable across the TeX community. It is not a book that most people will read cover to cover. Rather, you will dip into it to find specific information. That's how I reviewed the book. Specifically, I read the chapters on "PSTricks", "The world of color", "Using PostScript fonts", and "PostScript drivers and tools" quite thoroughly with good benefit. My principal criticism of the book is that the authors do not discuss installation of the various packages in enough detail to be useful. As anyone who has struggled to install one of these packages knows, getting them working correctly is no mean feat.

◇ David F. Rogers
817 Holly Drive East
Route 10 Amberley
Annapolis MD 21401
dfr@usna.navy.mil

Hints and Tricks

'Hey — It Works!'

Jeremy Gibbons

Welcome to *Hey — it Works!*. This column is devoted to interesting, elegant or just surprising tips and tricks for (LA)TEX. The title has two interpretations: surprise — 'well, bless my cotton socks, it actually works!' — and pragmatism — 'don't knock it, it does the trick'. Articles fitting either interpretation are welcome, whether arcane wizardry or simple but useful techniques; the overriding criterion is brevity and elegance.

This column ran in *TEX and TUG News* until the demise of that newsletter in 1995. Barbara Beeton has kindly agreed to continue it in *TUGboat*, which has absorbed the newsletter. During the intervening period, I have moved from New Zealand to the UK. I have also collected all back-issues of the column since I took it over from Christina Thiele in 1993, and made them available through the URL

`http://www.brookes.ac.uk/~p0071749/hiw/`

In this issue we have three contributions. Donald Arseneau, as ever, has a nice piece showing how to remove a counter from the list of counters to be reset at the start of each section-like unit in LATEX; this is more difficult than adding a counter, but

Donald presents a very elegant solution. Ramón Casares shows how to change the default thickness for `\hrule`s and `\vrule`s from the standard hardwired `0.4pt`. The final article is by yours truly, and explains how to define a 'small verbatim' environment, as used in this column.

◇ Jeremy Gibbons
School of Computing and Mathematical Sciences
Oxford Brookes University
Gipsy Lane, Headington
Oxford, OX3 0BP, UK
jgibbons@brookes.ac.uk
URL: http://www.brookes.ac.uk/~p0071749/

1 Removing a counter from a reset list

By default, the LaTeX `report` and `book` classes reset equation and other numbers at the start of each chapter, but once I needed to number equations (figures, etc.) sequentially throughout a report. I could have created an entire document class (copying from `report.cls`) but with the counters defined differently, reducing `\newcounter{equation}[chapter]` to just `\newcounter{equation}`, and likewise for other counters. That's a bit ridiculous, though, when it is the only change I want to make! What is more convenient is to *remove* the counter resets.

LaTeX keeps the list of counters that are to be reset with each section-like ⟨*unit*⟩ (where ⟨*unit*⟩ is "chapter," "section," etc.) in the macro `\cl@`⟨*unit*⟩. Clearly, what I needed was to remove the equation counter from the list `\cl@chapter`. There is a simple (internal) LaTeX macro called `\@addtoreset` to add a counter to this list, but there is none for removal; so I wrote one of the form

```
\@removefromreset{equation}{chapter}
```

Johannes Braams wrote on this topic in *TUGboat* Vol. 15 (Dec. 1994, p. 496) and explains the functioning of the reset list, but his solution is more complex than necessary, with nested looping, whereas the following definition efficiently redefines the list in a single scan.

```
\def\@removefromreset#1#2{%
 % preserve \@elt (probably unnecessary)
  \let\@tempb\@elt
 % put what we want to remove in \@tempa:
  \expandafter\let\expandafter\@tempa
    \csname c@#1\endcsname
 % define \@elt to check for removal:
  \def\@elt##1{\expandafter\ifx
    \csname c@##1\endcsname\@tempa\else
 % else, reinsert without execution:
    \noexpand\@elt{##1}\fi}%
 % redefine list as itself with removal:
  \expandafter\protected@edef
    \csname cl@#2\endcsname
    {\csname cl@#2\endcsname}%
 % restore \@elt
  \let\@elt\@tempb}
```

How does this work? The list `\cl@`⟨*unit*⟩ is a sequence of commands of the form `\@elt{`⟨*ctr*⟩`}`, one for each counter ⟨*ctr*⟩ (equation, e.g.) to be reset at the start of each ⟨*unit*⟩ (chapter). The command `\@removefromreset{equation}{chapter}` temporarily defines `\@elt{equation}` to disappear (i.e., to expand to nothing) but `\@elt{section}` to remain unchanged (i.e., to expand to itself). Then `\cl@chapter` is simply defined as itself, with these definitions in effect!

When numbering in a continuous sequence, I don't like the chapter number as a prefix to the equation numbers, so I redefine the equation numbering with

```
\def\theequation{\arabic{equation}}
\@removefromreset{equation}{chapter}
```

along with removals for the figure and table counters, if desired. These lines and the definition of `\@removefromreset` should be put in a style file. The definition of `\@removefromreset` is on CTAN in the file `macros/latex/contrib/other/fragments/removefr.tex`, which you should copy into any local `.sty` or `.cls` file where you would like to use `\@removefromreset`.

◇ Donald Arseneau
TRIUMF
4004 Wesbrook Mall,
Vancouver B.C.
Canada V6T 2A3
asnd@reg.triumf.ca

2 Default Rule Thickness

Sometimes the default rule thickness, `0.4pt`, is not the one you need. The solution is to write, for example, `\vrule width 1pt` instead of `\vrule`. But this does not work if the `\vrule`s are hidden inside macros you do not want to modify. When I was in such a situation my first reaction was to look for a ⟨*dimen parameter*⟩ in *The TeXbook*, p. 274. As usual in these cases, I saw some parameters I had never imagined, but nothing resembling the `\rulethickness` I needed[1].

Fortunately in TeX (almost) everything can be done.

```
\let\oldhrule=\hrule
```

[1] There is a `\fontdimen` in maths fonts controlling default rule thickness, but that applies only to rules in maths mode. –jg

```
\let\oldvrule=\vrule
\def\rulethickness{\afterassignment
 \dorulethickness\dimen0 }
\def\dorulethickness{\edef\hrule
  {\oldhrule height\dimen0 }%
 \edef\vrule
  {\oldvrule width\dimen0 }}
```

From now on a declaration as `\rulethickness=1pt` makes the default thickness of all rules equal to `1pt`. The next line is an example of a `1pt \hrule`.

Note that now, if you want a `0.4pt` rule you have to write `\vrule width 0.4pt` as expected. Note also that you could omit the '`=`' in the assignment; `\rulethickness 1pt` or `\rulethickness1pt` are also valid.

◇ Ramón Casares
Telefónica de España
rcg@tid.es

3 Small verbatim material

In order to keep verbatim material (such as the code for macros in this column, or example programs on OHP slides) to a reasonable length, it is often desirable to set it in `\small` size rather than the normal size. What you *can't* do to achieve this is to define a `smallverbatim` environment by

```
\newenvironment{smallverbatim}
  {\small\verbatim}{\endverbatim}
```

— an approach that would work if the task were instead to define, say, a 'small quotation environment'. The `verbatim` environment is a strange beast, quite unlike other environments; it is not ended by 'executing' `\end{...}` as other environments are, but rather by finding *exactly* the 14 characters '`\end{verbatim}`' in the file. Finding a macro that expands to these characters is not enough. (The perils of macro expansion languages!)

One apparent solution is to forgo the specialized environment, and do it manually each time:

```
Here is the previous paragraph.
\begin{small}\begin{verbatim}
Verbatim material.
\end{verbatim}\end{small}
Here is the next paragraph.
```

Unfortunately, this doesn't work well. By the time the `\begin{verbatim}` has ended the previous paragraph, the size has already been set to `\small` and the `\baselineskip` reduced accordingly; the result is that the entire previous paragraph gets set with too little leading. Here is an example, using `\scriptsize` verbatim and a paragraph without ascenders or descenders to emphasize the effect:

Here is tHe Previous ParaGraPH. Here is tHe Previous ParaGraPH. Here is tHe Previous ParaGraPH.

```
Here is some
verbatim material.
```

Here is tHe next ParaGraPH. Here is tHe next ParaGraPH. Here is tHe next ParaGraPH.

If you look carefully, you will find many published papers and books exhibiting this problem.

You could avoid this scrunching up by leaving a blank line (or a `\par`) before the `\begin{small}`, thereby ending the previous paragraph before the size change. However, it is all too easy to forget that blank line, making the document rather fragile (a blank line before a `verbatim` environment appears to be ignored under normal circumstances). A better solution than this is needed.

For the earlier editions of this column, which appeared in *TEX and TUG News* and used the old LaTeX 2.09, I had to resort to writing my own environment, mimicking the standard `verbatim` environment but changing to `small` size between ending the previous paragraph and starting the verbatim material. However, LaTeX 2ε provides a very convenient hook for just such a change: the macro `\verbatim@font`. The default definition is

```
\def\verbatim@font{\normalfont\ttfamily}
```

but you can make all your verbatim material small by replacing the `\normalfont` by `\small`. This, however, has the unfortunate side effect of making `\verb` material also appear small, which may not be what you want. An effective solution can be obtained by redefining the `verbatim` environment so that it changes `\verbatim@font` just for that single instance of the environment:

```
\let\VERBATIM\verbatim
\def\verbatim{%
  \def\verbatim@font{\small\ttfamily}%
  \VERBATIM}
```

which is what I have done for this column.

◇ Jeremy Gibbons
Oxford Brookes University
jgibbons@brookes.ac.uk

Software and Tools

The pdfTeX user manual

Han The Thanh and Sebastian Rahtz

1 Introduction

The pdfTeX project started in 1996 at the Faculty of Informatics, Masaryk University, Brno, Czech Republic. It forms a primary part of the MSc and PhD research of Han The Thanh, under the supervision of Jiří Zlatuška and Petr Sojka. The main purpose of the project was to create an extension of TeX that could create PDF directly from TeX and improve/enhance the result of TeX typesetting with the help of PDF. pdfTeX contains TeX as a subset. When PDF output is not set, it produces normal DVI output; when PDF output is selected, pdfTeX produces PDF output that looks identical to the DVI output. The next stage of the project, apart from fixing any errors in the program, is to investigate alternative justification algorithms, possibly making use of multiple master fonts.

pdfTeX is based on the original TeX sources and web2c, and has been successfully compiled on Unix, Amiga, Win32 and DOS systems. It is still under beta development and all features are liable to change.

This manual was compiled by Sebastian Rahtz from notes and examples by Han The Thanh. Many thanks are due to members of the pdfTeX mailing list (most notably Hans Hagen), whose questions and answers have contributed much to this manual.

2 Implementation details

The implementation of pdfTeX consists of two parts: the changes to TeX, and the addition of 'driver' features. The TeX-specific part is written as a change file to the original source of TeX and contains:

- a part that generates PDF code visually equivalent to DVI commands;
- virtual font processing;
- implementation of new primitives for PDF control.

The 'driver' part is written as several source files in C, and implement

- font mapping,
- font inclusion and partial downloading,
- font re-encoding,
- PNG picture insertion (using the public domain `libpng` code),
- text compression (using the public domain `zlib` code).

2.1 Compilation

pdfTeX is supplied as a set of additions to the standard Unix Web2c setup, and is a standard part of Web2c 7.2 (from January 1998); compilation should present no problems on most Unix systems. Porting it to other setups will require a little work, because of the requirement to combine the normal TeX Web output, and the parts written in C.

For Win32 systems (Windows 95, Windows NT) there are two packages that contain pdfTeX, both in `CTAN:systems/win32`. Web2c for Win32 is maintained by Fabrice Popineau (`mailto:popineau@ese-metz.fr`), and MikTeX by Christian Schenk (`mailto:cschenk@berlin.snafu.de`).

A binary version of pdfTeX for Amiga is made available (`CTAN:systems/\pdftex/bin/Amiga`) by Andreas Scherer (`mailto:Scherer@Physik.RWTH-Aachen.De`).

To make the situation more complicated, a Win32 binary version of pdfTeX compiled with Cygnus tools will also be made available in `CTAN:systems/pdftex/bin/Win32`. This version will be compiled in the same way as Unix systems.

2.2 Search paths

When runing pdfTeX, some extra search paths need to be set up beyond those normally requested by TeX itself; in web2c these are:

VFFONTS the path where pdfTEX looks for virtual fonts.

T1FONTS the path where pdfTEX looks for Type1 and TrueType fonts.

TEXPSHEADERS the path where pdfTEX looks for the font mapping file (`pdftex.map`), PNG pictures and encoding files (`*.enc`).

3 Fonts

pdfTEX can only work with Type 1 and TrueType fonts at present, and a source must be available for all fonts used in the document, except for the 14 base fonts supplied by Acrobat Reader (Times, Courier, and Symbol). It is *not* possible to use METAFONT-generated fonts in pdfTEX—apart from the technical reasons, the resulting Type 3 fonts render very poorly in Acrobat Reader. Given the free availability of Type 1 versions of all the Computer Modern fonts, and the ability to use standard PostScript fonts without further ado, most existing TEX users should be able to experiment with pdfTEX.

pdfTEX reads a map file called `pdftex.map`. Reencoding and partial downloading for each font is specified in this file. Every font must be listed in this file, each on a separate line. The syntax of each line is similiar to dvips' map files (but may be changed later). Each line contains the following fields (some of them are optional):

`texname basename fontflags fontfile encoding`

`texname`: the name of the TFM file

`basename`: the base font name (PostScript font name)

`fontflags`: an integer number specifying the most important characteristics of the font. This number is significant only in the case that the font file is *not* included, and Acrobat Reader is asked to simulate missing font using its multiple master defaults. It is a PDF font descriptor (see PDF manual, section 7.9.2) which is a 32-bit integer containing a collection of Boolean attributes, with bit 1 being the least significant. Attributes are true if the corresponding bit is set to 1 in the integer. The meanings of the bits is as follows:

1	Fixed-width font
2	Serif font
3	Symbolic font
4	Script font
5	(reserved)
6	Uses the Adobe Standard Roman Character Set
7	Italic
8–16	(reserved)
17	All-cap font
18	Small-cap font
19	Force bold at small text sizes
20–32	(reserved)

Bit 6 indicates that the font's character set is the Adobe Standard Roman Character Set, or a subset of that, and that it uses the standard names for those characters. About bit 19, the PDF specification says

> ... used to determine whether or not bold characters are drawn with extra pixels even at very small text sizes. Typically, when characters are drawn at small sizes on very low resolution devices such as display screens, features of bold characters may appear only one pixel wide. Because this is the minimum feature width on a pixel-based device, ordinary non-bold characters also appear with one-pixel wide features, and cannot be distinguished from bold characters. If bit 19 is set, features of bold characters may be thickened at small text sizes.

It should be stressed that the font flags provided for many fonts in the currently distributed `pdftex.map` are not correctly derived.

`fontfile`: the name of the font source file. This must be a Type 1 or TrueType font file. If it is preceded by a `<` then the font file will be partly downloaded; if it preceded by a double `<<` then the font file will

be included entirely. Otherwise the font is not included, and only the font parameters are extracted. Note that the font file *must* always be available at runtime, even if it not downloaded.

`encoding`: the encoding vector used for the font. It may be preceded by a `<`, but the effect is the same. The format is identical to that used by dvips.

Here are some sample lines from `pdftex.map`:

- include font entirely without reencoding

  `pplr8r Palatino-Roman 34 <<Palatino-Roman.pfb`

- include font partly without reencoding

  `pplr8r Palatino-Roman 34 <Palatino-Roman.pfb`

- do not include font, or reencode it, but extract parameters from font file

  `pplr8r Palatino-Roman 34 Palatino-Roman.pfb`

- include font entirely and reencode

  `pplr8r Palatino-Roman 34 <<Palatino-Roman.pfb 8r.enc`

- partially include font and reencode

  `pplr8r Palatino-Roman 34 <Palatino-Roman.pfb 8r.enc`

- do not include font but extract parameters from font file and reencode

  `pplr8r Palatino-Roman 34 Palatino-Roman.pfb 8r.enc`

- A TrueType font can be used in the same way as a Type 1 font

  `Cxr10 Cxr10 34 <Cxr10.ttf Cxtext.enc`

- base font can be also reencoded; the name of the font file is simply ignored

  `phvr8r Helvetica 32 <Helvetica.pfb 8r.enc`

4 New primitives

Here is a short description of new primitives added by pdfTEX:

`\pdfoutput`=*n*

Integer parameter specifying whether the output format should be DVI or PDF. Positive value means PDF output, otherwise DVI output. This parameter cannot be specified *after* shipping out the first page.

`\pdfcompresslevel`=*0*

Integer parameter specifying the level of text compression via `zlib`. Zero means no compression, 1 means fastest, 9 means best, 2..8 means something in between.

`\pdfpagewidth`=*dimen*

`\pdfpageheight`=*dimen*

Dimension parameters specifying the page width and page height of PDF output. If not specified then they are calculated as `\hsize` or `\vsize` + 2 × (1*in* + `\hoffset` or `\voffset`), when pdfTEX writes the first page.

`\pdfliteral`{*pdf text*}

Like `\special`, this can be used to insert raw PDF code into the output. This allows support of color, and text transformation.

`\pdfinfo` *`author`*{*author*} *`title`*{*title*} *`subject`*{*subject*} *`keywords`*{*keywords*}

Specify document information. All keys are optional; if this information is provided, it can be seen in Acrobat Reader with the menu option Document Information ⇝ General

`\pdfcatalog` *pagemode*`{`*pagemode*`}` *uri*`{`*uri*`}`

Document display information. Both keys are optional. *URI* provides the base URL of the document, and *pagemode* determines how Acrobat displays the document on startup. The possibilities are:

/UseNone Open document with neither outline nor thumbnails visible.

/UseOutlines Open document with outline visible.

/UseThumbs Open document with thumbnails visible.

/FullScreen Open document in full-screen mode. In full-screen mode, there is no menu bar, window controls, nor any other window present.

The default is **/UseNone**.

`\pdfimage` *height height width width filename*

Insert a bitmap image in PNG format, optionally changing width, height or both. Dimensions which are not specified will be treated as zero. If both height and width are zero then the box takes the natural size of the image. If one of them (width or height) is zero and the second is not, then the first one (the zero one) will be set to a value proportional to the second one so to make the box have the same proportion of width and height as the image natural size. If both width and height are positive then the image will be scaled to fit these dimensions.

`\pdfform` *num*

Write out the TEX box *num* as a Form object to PDF output.

`\pdflastform`

Returns the object number of the last Form written to the PDF file

`\pdfrefform \name`

Put in a reference to the PDF Form called `\name`.

These macros support a feature called "object reuse" in pdfTEX. The idea is to create a Form object in PDF. The content of the Form object corresponds to the content of a TEX box, which can contain pictures and references to other Form objects as well. After that the Form can be used by simply referring to its object number. This feature can be useful for large document with a lot of similiar elements, as it can reduce the duplication of identical objects.

`\pdfannottext` *width width height height depth depth attr*`{`*attr*`}` `{`*text*`}`

Attach a text annotation at the current point in the text. The attributes can be used to specify the default appearance of the annotation (e.g., `/Open true` or `/Open false`), as well as many other features (see Table 6.8 of the PDF manual).

`\pdfdest` *num num name*`{`*name*`}` *appearance*

Establish a destination for links and bookmark outlines; the link must be identified by either a number or a symbolic name, and the way Acrobat is to display the page must be specified; *appearance* must be one of

fit	fit whole page in window
fith	fit whole width of page
fitv	fit whole height of page
fitb	fit whole 'Bounding Box' page
fitbh	fit whole width of 'Bounding Box' of page
fitbv	fit whole height of 'Bounding Box; of page

`\pdfannotlink` *height height depth depth attr*`{`*attr*`}` *action*

Start a hypertext link; if the optional dimensions are not specified, they will be calculated from the box containing the link. The *attributes* (explained in great detail in section 6.6 of the PDF manual) determine the appearance of the link. Typically, this is used to specify the color and thickness of any border around the link. This `/C [0.9 0 0] /Border [0 0 2]` specifies a color (in RGB) of dark red, and a border thickness of 2 points.

The *action* can do many things; some possibilities are

goto num *n*	
goto name {*refname*}	Jump to a point established as *name*with `\pdfdest`
goto file {*filename*}	Open a local file; this can be used *with* a *name* or *num* specification, to point to a specific location on the file. Thus `goto file{foo.pdf} name{intro}`
thread num {*n*}	
thread name {*refname*}	Jump to thread identified by *n* or *refname*
user {*spec*}	Perform user-specified action. Section 6.9 of the PDF manual explains the possibilities. A typical use of this is to specify a URL, as `/S /URI /URI (http://www.tug.org/)`.

`\pdfendlink`

Ends link; all text between `\pdfannotlink` and `\pdfendlink` will be treated as part of this link. pdfTEX may break the result across lines (or pages), in which case it will make several links with the same content.

`\pdfoutline` *action* **`count`** *count* {*text*}

Create a outline (or bookmark) entry. The first parameter specificies the action to be taken, and is the same as that allowed for `\pdfannotlink` (see above, though note that the `Page` key does not work properly at present). The *count* specifies the number of direct subentries under this entry, 0 if this entry has no subentries (in this case it may be omitted). If the number is negative, then all subentries will be closed and the absolute value of this number specifies the number of subentries. The *text* is what will be shown in the outline window (note that this is limited to characters in the PDFEncoding vector).

`\pdfthread` ***num*** *num* ***name***{*name*}

Start an article thread; the corresponding `\pdfendthread` must be in the box in the same depth as the box containing `\pdfthread`. All boxes in this depth level will be treated as part of this thread. An identifier (*n* or *refname*) must be specified; threads with same identifier will be joined together.

`\pdfendthread`

Finish the current thread.

`\pdfthreadhoffset`=*dimen*

`\pdfthreadvoffset`=*dimen*

Specify thread margins.

One way to learn more about how to use these primitives is to have a look at the file `example.tex` in pdfTEX distribution.

5 Graphics and color

Probably the biggest single usage problem with pdfTEX at the present time is the inclusion of graphics. The program only directly supports graphic inclusion in one bitmap format, PNG (Portable Network Graphics).

Two commonly-used techniques are not available — the inclusion of Encapsulated PostScript figures, and the inclusion of raw PostScript commands (the techique utilized by the pstricks package). Although PDF is a direct descendant of PostScript, it lacks any programming language commands, and cannot deal with arbitrary PostScript. There are two ways to proceed with existing EPS files: firstly, convert them to

PNG (using programs like Image Magick, Image Alchemy, or Ghostscript); or secondly, try converting them to simple PDF. If the picture has no special fonts, the chances are quite good that Ghostscript's pdf writer will produce a file containing a single PDF object, which can be included using `\pdfliteral` commands (this is managed by the standard LaTeX graphics package).

Other alternatives for graphics in pdfTeX are:

LaTeX picture mode Since this is implemented simply in terms of font characters, it works in exactly the same way as usual;

Xy-pic If the PostScript backend is not requested, Xy-pic uses its own Type 1 fonts, and needs no special attention;

tpic The 'tpic' `\special` commands (used in some macro packages) can be redefined to produce literal PDF, using macros by Hans Hagen;

MetaPost Although the output of MetaPost is PostScript, it is in a highly simplified form, and a MetaPost to PDF conversion (written by Hans Hagen and Tanmoy Bhattacharya) is implemented as a set of macros which read MetaPost output and support all of its features;

It is possible to insert a "pure" PDF file (PDF that has only one page without fonts, bitmap and other resources) with a macro package that reads the external PDF file line by line.

The two latter macro packages are part of ConTeXt (`supp-pdf.tex` and `supp-mis.tex`), but also work with LaTeX and may be distributed separately.

For new work, the MetaPost route is highly recommended. For the future, Adobe has announced that they will define a specification for 'encapsulated PDF', and this should solve some of the present difficulties.

6 Macro packages supporting pdfTeX

- For LaTeX users, Sebastian Rahtz' hyperref package has substantial support for pdfTeX, and provides access to most of its features. In the simplest case, the user merely needs to load hyperref with a 'pdftex' option, and all cross-references will be converted to PDF hypertext links. PDF output is automatically selected, text compression turned on, and the page size is set up correctly. Bookmarks are created to match the table of contents.
- The standard LaTeX graphics and color packages have `pdftex` options, which allow use of normal color, text rotation, and graphics inclusion commands. Only PNG and MetaPost files can be included.
- The ConTeXt macro package by Hans Hagen (`mailto:pragma@pi.net`) has very full support for pdfTeX in its generalized hypertext features.
- Hypertexted PDF from `texinfo` documents can be created with `pdftexinfo.tex`, which is a slight modification of the standard `texinfo` macros. This is part of the pdfTeX distribution.
- A similiar modification of the `webmac`, called `pdfwebmac.tex`, allows production of hypertexted PDF versions of programs written in WEB. This is part of the pdfTeX distribution.

Some nice samples of pdfTeX output can be found on the TUG Web server, at `http://www.tug.org/applications/pdftex/`.

◇ Han The Thanh
Faculty of Informatics
Masaryk University
Brno, Czech Republic
`thanh@informatics.muni.cz`

◇ Sebastian Rahtz
7 Stratfield Road
Oxford OX2 7BG
United Kingdom
`s.rahtz@elsevier.co.uk`

Spindex — Indexing with Special Characters

Laurence Finston

1. Introduction

Books in the field of philology, among others, often contain many special characters: letters like ð and þ, ligatures like æ and œ, phonetic symbols like ʃ and ŋ and even more unusual ones. If these books require indexes, words with these special characters must be sorted alphabetically. However, to the best of my knowledge, the available indexing programs are only able to sort words in English, or at best in a handful of European languages. Spindex (for "Special Index") is a package that can sort arbitrary special characters alphabetically. It can also be adapted for use with languages that do not use the Latin alphabet.

TEX has no built-in routines for alphabetical sorting, so it is necessary to use the sorting routines belonging to the operating system, a programming language, or another program. Spindex is a combination of TEX macros in the file `spindex.tex` and a program written in Common Lisp in the file `spindex.lsp`. It is intended for use with `plain` TEX, but it is possible (with some difficulty) to use it with LATEX, too.

The first section of this article explains Spindex for the user who just wants to use it for making an index, and doesn't care about how it works. The following section explains some of the principles behind the TEX macros and the Lisp program.

2. Using Spindex

In order to use Spindex, the file `spindex.lsp`, containing the Lisp program, must be in your working directory, and `spindex.tex`, containing the definition of the TEX macro `\indexentry` and additional TEX code, must be either in your working directory or in a directory in TEX's load path as defined in your `texmf.cnf` file (if you don't know what this is, ask your local TEX wizard, or just put the file in your working directory). Your input file must include the line `\input spindex` before you use `\indexentry` for the first time.

When you use `\indexentry`, it causes TEX to write a file of Lisp code called `entries.lsp`. When TEX is done with your input file, you invoke the Lisp interpreter and give `spindex.lsp` to it as input. If you're using the Gnu Lisp interpreter, which is what I use, you type

```
gcl<spindex.lsp
```

The program in `spindex.lsp` loads `entries.lsp` and creates a TEX file containing the index called `index.tex`. Now you can run TEX on `index.tex`. Below I describe how to automate this process and include `index.tex` in your original input file.

2.1. The macro `\indexentry`. An index entry is created using `\indexentry`, which has six arguments, all of which except for `#1` may be empty (i.e., `{}`). TEX does not have true optional arguments, but it is possible to define macros so that they check whether an argument is empty or not, simulating the effect of optional arguments. The consequence of this is that six sets of braces must always follow `\indexentry` whether there's anything in them or not.

The first argument, `#1`, is `name`, which is used for alphabetizing the entries, and it is usually what is written to the index. It is the only required argument. An occurrence of `\indexentry` with only the `name` argument is the simplest possible kind. For example,

```
\indexentry{nouns}{}{}{}{}{}
```

on page 54

$\Longrightarrow$

nouns . . . . . 54.

In most cases, `\indexentry` will be typed into the input file directly after the word or phrase that it refers to:

```
a noun\indexentry{nouns}{}{}{}{}{} is
a word that refers ...
```

produces the following output:

a noun is a word that refers ...

Putting `\indexentry` directly after the word or phrase it refers to prevents a page break between them, which would cause an incorrect page number to appear in the index. However, `\indexentry` can also stand alone, as in the examples below. Note that `\indexentry` has no effect on the output file. All it does is write information to `entries.lsp`, which is used for making the index. However, I use a conditional called `\ifdraft` for editing purposes that makes `\indexentry` write a marginal hack whenever `\drafttrue`, i.e., whenever `\ifdraft` expands to `\iftrue`.

a noun is a word that refers ... `*nouns*`

For the final draft, I set `\draftfalse`, and the marginal hacks disappear.

Argument `#2` is `text`, and will usually be empty. If it's not empty, it's what's written to the index, but the entry is still alphabetized according to `name`.

```
\indexentry{A}{A (the letter A)}{}{}{}{}
```

$\Longrightarrow$

A (the letter A) . . . . . 96.

but "␣(the letter A)" does not affect the alphabetization of the entry.

The `text` argument can also be used for putting comments into the index at a particular place.

```
\indexentry{nouns}{*Comment*}{}{}{}{}
\indexentry{prepositions}{}{}{}{}{}
\indexentry{adverbs}{}{}{}{}{}
```

$\Longrightarrow$

adverbs . . . . . 87.
Comment . . . . . 87.
prepositions . . . . . 87.

Note that "*Comment*" is put where "nouns" would go. The `text` argument only has an effect when an entry is created. After that it's ignored, so if you want a `text`, you must make sure it's set the first time. It would be easy to change this, but I felt that it was safer to program it this way. Most of the time `text` will not be used. It is only for special cases like these.

The best way to set `text` is to use dummy entries at the beginning of your input file where the page number is suppressed using argument `#3`. A comment, like the one in the previous example, also shouldn't have a page number and leaders attached. Suppressing the page number can also be useful for editing, when you're not sure whether to include a particular occurrence of an entry in the index. It doesn't matter what appears in `#3`; if it's non-empty, this occurrence of `\indexentry` will not cause the current page number to be added to the list of page numbers for this entry.

```
\indexentry{verbs}{}{np}{}{}{}
```

$\Longrightarrow$

verbs

I like to use "np" (for "no page") in `#3`, but it can be anything within reason.[1] If an entry has no page numbers, no leaders are printed. Suppressing the page number in one invocation of `\indexentry` doesn't affect another invocation on the same page.

```
\indexentry{verbs}{}{np}{}{}{}
\indexentry{verbs}{}{}{}{}{}
```

$\Longrightarrow$

verbs . . . . . 123.

Argument `#4` is for a cross-reference. A cross-reference can be an arbitrary string or it can correspond to another entry. Here's an entry with a cross-reference that refers to an arbitrary string.

```
\indexentry{ships}{}{}{transport}{}{}
```

$\Longrightarrow$

ships . . . . . 75.
See also: transport

Here's one with a cross-reference that refers to another entry.

```
\indexentry{ships}{}{}{boats}{}{}
\indexentry{boats}{}{}{}{}{}
```

$\Longrightarrow$

boats . . . . . 54.
ships . . . . . 54.
See also: boats

Doesn't look much different, does it? But when a cross-reference refers to an entry that had a `text` (`#2`) argument, there is a difference.

```
\indexentry{boats}%
     {boats (lat. naves)}{}{}{}{}
\indexentry{ships}{}{}{boats}{}{}
```

$\Longrightarrow$

boats (lat. naves) . . . . . 54.
ships . . . . . 54.
See also: boats (lat. naves)

The cross-reference uses the `text` of an entry, if it exists. If there are multiple cross-references, they are alphabetized according to what is actually printed, i.e., the `text`s, if they exist, whereas the entries in the index are always alphabetized according to `name`.

Spindex allows 3 levels of nesting – headings, subheadings and subsubheadings. Argument `#5` is the `heading`, if the entry is a subheading or a subsubheading, and `#6` is the `subheading`, if the entry is a subsubheading. This is how you make a subheading entry:

```
\indexentry{transitive}{}{}{}{verbs}{}
```

$\Longrightarrow$

verbs
  transitive . . . . . 54.

Here's one for a subsubheading entry:

```
\indexentry{active}{}{}{}{verbs}%
     {transitive}
```

$\Longrightarrow$

verbs
  transitive
    active . . . . . 49.

[1] An undefined control sequence or a macro with insufficient arguments will cause an error.

A subheading or subsubheading entry will create an entry for its `heading` and/or `subheading`, if these don't already exist.

Here's a slightly tricky example (the line `\hbox{}\eject` is only there to end page 57).

```
\pageno=57
\indexentry{monosyllabic}{}{}{}%
     {adverbs}{temporal}
\hbox{}\eject
\indexentry{adverbs}{sbrevda}{}{}{}{}
```

$\Longrightarrow$

adverbs . . . . . 58.
temporal
monosyllabic . . . . . 57.

Do you see why "sbrevda" is not written to the index? The first invocation of `\indexentry`, for "adverbs, temporal, monosyllabic", caused entries for "adverbs" and "adverbs, temporal" to be created automatically. When `\indexentry` was invoked for "adverbs" in its own right, on page 58, the text argument was ignored, because the entry for "adverbs" had already been created. The best way to deal with this problem is by using a dummy entry, like this:

```
\pageno=1
\indexentry{adverbs}{sbrevda}{x}{}{}{}
\hbox{}\eject
\pageno=57
\indexentry{monosyllabic}{}{}{}%
     {adverbs}{temporal}
\hbox{}\eject
\indexentry{adverbs}{}{}{}{}{}
```

$\Longrightarrow$

sbrevda . . . . . 58.
temporal
monosyllabic . . . . . 57.

Here I use "x" to suppress the page number for the dummy entry. Subsequent invocations of `\indexentry` for "adverbs", like the one on page 58, needn't specify the `text` argument, since it's ignored.

Sometimes it might be desirable to put sub- or subheadings in order, but not in alphabetical order, if another ordering principle seems more appropriate.

```
\pageno=1
\indexentry{light}{light, visible}%
     {xxx}{}{}{}
\indexentry{wavelengths}{}{}{}%
     {light}{}
\hbox{}\eject
\indexentry{f}{violet}{}{}{light}%
     {wavelengths}
\hbox{}\eject
\indexentry{d}{green}{}{}{light}%
     {wavelengths}
\hbox{}\eject
\indexentry{b}{orange}{}{}{light}%
     {wavelengths}
\hbox{}\eject
\indexentry{c}{yellow}{}{}{light}%
     {wavelengths}
\hbox{}\eject
\indexentry{light}{}{}{}{}{}
\hbox{}\eject
\indexentry{a}{red}{}{}{light}%
     {wavelengths}
\hbox{}\eject
\indexentry{e}{blue}{}{}{light}%
     {wavelengths}
```

$\Longrightarrow$

light, visible . . . . . 6.
wavelengths . . . . . 1.
red . . . . . 7.
orange . . . . . 4.
yellow . . . . . 5.
green . . . . . 3.
blue . . . . . 8.
violet . . . . . 2.

The subsubsubheadings (the colors of visible light) are alphabetized according to their `name`s, i.e., "a", "b", "c", etc. This has the effect of putting them in order according to their wavelengths. Since there are no other subsubheadings, this causes no problems. Some items may have a conventional order that takes precedence over the alphabet.

```
\indexentry{Bears, the Three}{}{}%
     {Goldilocks}{}{}
  \indexentry{c}{Baby}{}{}%
     {Bears, the Three}{}
  \indexentry{c}{Baby}{}{}%
     {Bears, the Three}{}
  \indexentry{a}{Papa}{}{}%
     {Bears, the Three}{}
  \indexentry{b}{Mama}{}{}%
     {Bears, the Three}{}
```

$\Longrightarrow$

Bears, the Three . . . . . 23.
See also: Goldilocks
Papa . . . . . 23.
Mama . . . . . 23.
Baby . . . . . 23.

Cross-references can refer to subheadings and subsubheadings, too:

```
\indexentry{schooners}{}{}{}{ships}%
     {sailing}
\indexentry{rigging}{}{}%
     {ships-sailing-schooners}%
     {}{}
```

⟹

rigging . . . . . 54.
See also: ships, sailing, schooners
ships
sailing
schooners . . . . . 54.

A cross-reference that refers to a heading entry simply uses the `name` argument from that entry.

```
\indexentry{carnivores}{}{}{mammals}{}{}
\indexentry{mammals}{}{}{}{}{}
```

⟹

carnivores . . . . . 25.
See also: mammals
mammals . . . . . 25.

It doesn't matter if the entry being used as a cross-reference has a `text`; you use the `name` anyway, but the `text` is printed to the index file.

```
\indexentry{fish}%
     {fish ({|it|pisces})}%
     {}{}{}{}
\indexentry{oceans}{}{}{fish}{}{}
```

⟹

fish (*pisces*) . . . . . 100.
oceans . . . . . 100.
See also: fish (*pisces*)

When a subheading entry is used as a cross-reference, its `heading` and `name` arguments, separated by a hyphen, are used in the `cross-reference` argument of the entry that refers to it.

```
\indexentry{wolves}{}{}{bears-brown}{}{}
\indexentry{brown}{}{}{}{bears}{}
```

⟹

bears
brown . . . . . 371.
wolves . . . . . 371.
See also: bears, brown

When a subsubheading entry is used as a cross-reference, its `heading`, `subheading` and `name` arguments, separated by hyphens, are used in the `cross-reference` argument of the entry that refers to it.

```
\indexentry{American}%
     {American (Eastern)}%
     {}{}{bears}{brown}
\indexentry{wolves}{}{}%
     {bears-brown-American}{}{}
```

⟹

bears
brown
American (Eastern) . . . . . 41.
wolves . . . . . 41.
See also: bears, brown, American (Eastern)

The syntax of cross-references is:

⟨cross-reference⟩ ⟶ ⟨arbitrary string⟩
| ⟨entry reference⟩
⟨entry reference⟩ ⟶ `heading`⟨suffix⟩
⟨suffix⟩ ⟶ ⟨empty⟩ | `-subheading`
| `-subheading-subsubheading`

Only one cross-reference can appear in any given occurrence of `\indexentry`.

Of course, subheading and subsubheading entries can themselves have cross-references, and their page numbers can be suppressed, too:

```
\indexentry{fish}{}{}{}{}{}
\indexentry{freshwater}{}{np}%
          {angling}{fish}{}
\indexentry{sturgeon}{}{}{caviar}%
          {fish}{freshwater}
```

⟹

fish . . . . . 14.
freshwater
See angling
sturgeon . . . . . 14.
See also caviar

So far, all of the examples have been of entries with only one page number. Here's an example with multiple page numbers.

```
\pageno=5
\indexentry{trains}{}{}{}{}{}
\hbox{}\eject
\pageno=10
\indexentry{trains}{}{}{}{}{}
\hbox{}\eject
\pageno=15
\indexentry{trains}{}{}{}{}{}
\hbox{}\eject
\pageno=25
\indexentry{trains}{}{}{}{}{}
\hbox{}\eject
```

⟹

trains . . . . . 5, 10, 15, 25.

If an entry occurs on consecutive pages, page ranges are printed to the index instead of the individual page numbers.

trains
diesel . . . . . 62–98.

electric . . . . . 105–210.
steam . . . . . 5–10.

Sometimes, the last number in a page range is abbreviated.

ships . . . . . 104–23.
sailing . . . . . 1004–200.
steam . . . . . 1239–98.

The rules for abbreviating page numbers are described on page 269.

If an entry has no page numbers, but it does have a cross-reference, "*See*" is printed instead of "*See also*".

```
\indexentry{adjectives}{}%
     {suppress page number!}%
     {pronouns}{}{}
```

⟹

adjectives
See pronouns

If there are two cross-references, they are separated by "*and*", and if there are three or more, the last two are separated by "*and*" and the others are separated with a semi-colon.

```
\indexentry{schooners}{}{}{}{ships}{}
\indexentry{ships}{}{}{boats}{}{}
\indexentry{ships}{}{}{transport}{}{}
\indexentry{ships}{}{}{fishery}{}{}
\indexentry{rigging}{}{}%
           {ships-schooners}{}{}
\indexentry{rigging}{}{}{boats}{}{}
```

⟹

rigging . . . . . 54.
See also: boats *and* ships, schooners
ships . . . . . 54.
See also boats; fishery *and* transport
schooners . . . . . 54.

If an entry has no page numbers, no cross-references and no sub- or subsubheadings, it will be printed to the index, but `spindex.lsp` will issue a warning.

If more than one index is desired, for instance an index of names and an index of subjects, it would not be difficult to add a seventh argument to indicate to which index an entry belongs.

2.2. Coding special characters and macros. By now, you're probably convinced that Spindex has plenty of bells and whistles, but the capabilities described so far don't offer any significant advantage over the available indexing packages. The real power of Spindex is its ability to perform alphabetical sorting on arbitrary special characters.

It is not possible to use the normal coding for special characters, like `\dh` for ð, `\th` for þ, `\ae` for æ, and `\o` for ø, in `\indexentry`'s arguments. If your computer can represent characters like "æ" on its screen, and you've defined `\catcode‘\æ=\active` and `\letæ=\ae`, you can't use "æ" in an `\indexentry` either. Nor can you use `~` as a tie. Instead, special characters are coded by leaving out the `\` and surrounding what remains with `||`, like this: `|dh|` for ð, `|th|` for þ, etc. Active characters, like `~`, if they are used in `\indexentry` at all, must use a similar coding using only non-active characters. To use special characters that are only available in math mode, just surround the coding with `$$`, e.g., `$|aleph|$`. Using `||` is actually better, since using the normal codings could result in a lot of nested braces, which would make the input file difficult to read, especially since `\indexentry` already has 6 sets of braces. (Incidentally, Spindex includes an Emacs-Lisp function for writing `\indexentry` which queries for the arguments and puts them inside the braces automatically.)

Here are some examples of using special characters in `\indexentry`.

```
\indexentry{|th|eir}{}{}{}{s|'a|}{}
\indexentry{s|ae|tninger}{}{}%
     {S|"a|tze}{}{}
\indexentry{$|aleph|$}%
     {$|aleph|$ --- The letter aleph}%
     {}{}{}{}
\indexentry{|poll|}%
     {|poll| -- Polish |poll|}%
     {}{}{}{}
```

⟹

ℵ — The letter aleph . . . . . 54.
ł – Polish ł . . . . . 54.
sá
þeir . . . . . 54.
sætninger . . . . . 54.
See also: Sätze

`||` can be used to code anything, in particular, any control sequence, not just special characters. For example:

```
\indexentry{{|it|verbs}}{}{}{}{}{}
```

⟹

verbs . . . . . 19.

You could achieve the same effect with

```
\indexentry{verbs}{{|it|verbs}}{}{}{}{}
```

but there is a difference. If

```
\indexentry{verbs}{}{}{}{}{}
```

and

```
\indexentry{{|it|verbs}}{}{}{}{}{}
```

were both used in an input file, they would create two different entries, printed on different lines, one in the current font (probably roman) and one in italic, but the entries would be identical with respect to alphabetization. Their order in the index file would correspond to the order of the invocations of `\indexentry` in the input file. In most cases, it will be easier to put a font change in the `text` argument, but in special circumstances it might be better to have it in the `name` argument instead.

2.2.1. Customizing `spindex.lsp`. There is a huge number of special characters available and each project will have its own special requirements. Even when the same characters are used, their order may differ. For these reasons, it is necessary for the user to customize `spindex.lsp` for each set of requirements. This is not difficult. In `spindex.lsp` you will find a list that looks like this.

```
(a b c d dh e f g h i j k l m
  n o p q r s t u v w x y z
  ae oslash acirc thorn)
```

These are the characters that will be assigned a unique integer value, in ascending order, for alphabetical sorting. The exact items in this list will depend on the user's requirements. A function called set-char-values assigns the integer values to variables with names based on the items in this list, i.e., a-value, b-value, . . . , thorn-value. Usually, more than one character will occupy the same position in the alphabet, so not all of the characters used will require their own value. Some share a value with a character in the list, for example, according to some alphabetization conventions, "á", "à", and "ā" will all use a-value. All of the uppercase letters share a value with their corresponding lowercase letters. In some languages, ligatures like "æ" and "œ" are treated as "ae" and "oe" respectively, so they are assigned a list of two values, i.e., (a-value e-value) and (o-value e-value). In Danish, however, "æ" has its own position toward the end of the alphabet, so if a user needs an index sorted according to Danish conventions, set-char-values will have to assign an integer value to a symbol for "æ".

Each ordinary character and special coding that may appear as an argument in `\indexentry` must be accounted for in the function letter-function in `spindex.lsp`. This is how the code in letter-function looks for an ordinary character:

```
((or (equal local-string "a")
     (equal local-string "A"))
 (setq current-int-list `(,a-value)))
```

Here's how the code looks for a special character:

```
((or (equal local-string "thorn")
     (equal local-string "th"))
 (setq current-int-list `(,thorn-value))
 (setq current-tex-code "{\th}"))
```

This tells `spindex.lsp` that `|th|` and `|thorn|` are valid special codings, that they are assigned the value thorn-value, and that they are to be replaced with `{\th}` when `spindex.lsp` writes the index file. Note that the names of the symbols need not correspond to the coding used in `\indexentry`: "þ" is coded as `\th` in TEX and can be coded as `|th|` or `|thorn|` in `\indexentry`. However, in the character list, the symbol associated with "þ" is called thorn. In other cases, the name of a symbol is not permitted to be the same as the coding in TEX and `\indexentry`. For instance, the coding for "ø" is `\o` and can be coded as `|o|` in `\indexentry`. However, the symbol in the character list may not be o, because this is already used for "o". So the symbol in the character list is called oslash. If a character like "ä", coded as `\"a` in TEX and `|"a|` in `\indexentry`, should be assigned its own value, the symbol name would have to be something like aumlaut instead of `"a`, since the `"` would cause a fatal error in `spindex.lsp`. Spindex includes detailed instructions for customizing the Lisp program.

2.3. Overview of `\indexentry`'s arguments

- Argument `#1` (`name`). Only required argument. Used for alphabetizing entries at all levels (heading, subheading and subsubheading). Printed to index file unless `#2` (`text`) is non-empty.
- Argument `#2` (`text`). Printed to index file if non-empty, but entry is alphabetized according to `name`. Also used when a cross-reference refers to this entry. Can be used for comments and other special purposes.
- Argument `#3` is used for suppressing the page number. Any string containing only characters of `\catcode`=11 ("letter") and/or `\catcode`=12 ("other") can be used safely.
- Argument `#4` (`cross-reference`). Can be an arbitrary string or refer to another entry at any level, using a special syntax described above. Entries at any level can have cross-references (see page 257).
- Argument `#5` (`heading`). Will be empty if the entry is a heading. If the entry is a subheading or a subsubheading, this argument refers to the heading entry, of which this entry is a sub- or subsubheading. Used for making a Lisp symbol.

- Argument `#6` (`subheading`). Will be empty if the entry is a heading or a subheading. If the entry is a subsubheading, this argument refers to the subheading entry, of which this entry is a subsubheading. Used for making a Lisp symbol.

2.4. Running Spindex. The `\indexentry` macro may write a marginal hack, but otherwise it has no effect on the file in which it is used. It simply writes a file of Lisp code that's used to generate another TEX file. Spindex does not in itself make any connection between the two TEX files. The user can (and must) decide what to do with them.

I use a combination of a UNIX shell script and a TEX driver file to control running TEX and Lisp. This is a rather complicated topic, since I also use them to control other things, like generating the table of contents, the bibliography, page references, etc. I plan on describing this technique in a subsequent article, but here is a simple example just for the index.

```
#### This is the shell script run_driver

if [[ -f index_switch.tex ]]
then
rm index_switch.tex
fi

tex driver

if [[ -f index_switch.tex ]]
then
gcl<"spindex.lsp"
tex driver
else
echo "There were no index entries"
fi
```

```
%%% This is the TeX driver file
%%% driver.tex

\newif\iffirstrun
\newread\indexin
\openin\indexin=index_switch
\ifeof\indexin
\firstruntrue
\else
\firstrunfalse
\let\suppressindex=t
\fi
\closein\indexin

\input spindex

\input input_file

\iffirstrun
\message{This is the first run,
     not inputting index}%
\else
\message{This is the second run,
     inputting index}%
\vfil\eject
\input index
\fi
\bye
```

The shell script `run_driver` runs TEX on the file `driver.tex`. If `\indexentry` isn't used, then `run_driver` is finished. Otherwise, it runs `spindex.lsp` to create the index file. Then it runs TEX on `driver.tex` again. This time, no file of Lisp code is written; instead, `driver.tex` inputs the index file and TEX exits.

2.5. "Faking" an index. Since `entries.lsp` and `index.tex` are both ordinary ASCII files, it's possible to edit them as one would edit any TEX file or Lisp program. Since they are automatically generated and old versions are overwritten, this would only make sense for polishing a final draft. But it is possible. More practical is a dummy TEX file that contains invocations of `\indexentry` but no text to be typeset, like the examples above. Explicit page breaks and numbering must be specified. This is an example of an index produced using a dummy file:

and this is the beginning of the dummy file that produced it:

```
%% This is dummy_index.tex
\input spindex
\input ipamacs
\font\ipatenrm=wsuipa10
\def\ipa{\ipatenrm}
\pageno=3
\indexentry{yyy}{This is a %
     comment where yyy would be.}%
     {np}{}{}{}
\indexentry{active}{active %
     (except deponentia)}%
     {}{nouns}{verbs}{transitive}
\hbox{}\eject
\pageno=122
\indexentry{active}{}{}%
     {|o|llebr|o||dh|}%
     {verbs}{transitive}
\hbox{}\eject
\pageno=121
\indexentry{active}{}{}{S|"a|tze}%
     {verbs}{transitive}
\hbox{}\eject
\pageno=120
\indexentry{active}{}{}{S|"a|tze}%
     {verbs}{transitive}
```

The complete dummy file contains a total of 73 `\indexentry` commands.

2.6. Getting Spindex. Spindex will be available on an ftp server under the normal conditions applying to free software. If you are interested, please contact me via email and I will tell you where to get it. The program `spindex.lsp` was written using the Gnu Lisp interpreter, which is free. The program itself should work without any trouble with a different Common Lisp interpreter; only two non-essential functions use the operating system interface, which always depends on the particular Lisp interpreter you're using. Getting these two functions to work with a different interpreter should require only minor adjustments.

3. Programming Spindex

3.1. Why not LaTeX? Spindex is designed for use with `plain` TeX. It's possible to use it with LaTeX, too, as mentioned above, but there are some difficulties involved. I find that LaTeX works well as long as one of its pre-defined formats can be used without significant changes. However, if modifications are necessary, I find that programming a format with `plain` TeX is much easier and gives better results. It's always a little risky to write macros when using a large package like LaTeX that already contains a lot of macros. In LaTeX especially, it's difficult to figure out exactly what macro or assignment is causing a certain effect, or even to understand the macro definitions. Many packages also change the `\catcode` of characters, which can cause serious problems. For instance, if you use a package that sets `\catcode‘\|=\active`, Spindex will fail.

The program in `spindex.lsp` functions independently of TeX or LaTeX and only one change is necessary to make `\indexentry` work in LaTeX: `\pageno` must be replaced by `\thepage`. The actual text of the index entries, the headings, subheadings, subsubheadings, page numbers and cross-references, will be the same whether you use TeX or LaTeX. However, `spindex.lsp` also writes formatting commands to the index file, and these must be compatible with the format and the output routine being used. The version of `spindex.lsp` that I'm making available writes formatting commands appropriate to the simple `plain` TeX format and output routine that are included in `spindex.tex`. The formatting is performed by a combination of the code written to `index.tex` by `spindex.lsp` and the definitions in `spindex.tex`. Since the formatting commands written to `index.tex` are defined in a general way, it's possible to make significant changes just by changing the definitions in `spindex.tex`, without making any changes to the Lisp program. However, if the user wants `spindex.lsp` to write different formatting commands, it's easy to modify it.

Using Spindex with LaTeX will require some experimentation to get it to produce the kind of formatting desired. Anyone who wishes to do this may feel free. There are many LaTeX formats and I rarely use any of them, so I have no interest in doing this experimenting. This is a task best left to a LaTeX programmer who really uses the formats.

3.2. Why Lisp? While it is possible to get TeX to jump through hoops, I usually find it easier to let TeX do what it does best, typesetting, and use

a conventional programming language for things like storing and manipulating data, alphabetizing, writing files, etc. While C seems to be the language of choice for front-end programs for TEX, Lisp offers a number of significant advantages, partly due to Lisp code being interpreted rather than compiled. It's possible to have TEX write executable Lisp code directly, so that it is unnecessary to write routines for reading data from files, and Lisp code is easier to test and debug than program code that must be compiled. Lisp also has many functions for sorting and manipulating strings and, of course, lists, Lisp's characteristic data type. In addition, the structure of the program in `spindex.lsp` depends on Lisp's ability to use undeclared variables, which is not possible in C. The program `spindex.lsp` is not very long, and it runs fast, at least on the installation I'm using (a Dec Alpha computer running Digital UNIX). I use the Gnu Lisp Interpreter, which is free and works well. Unfortunately, it does not conform to the newest standard described in Guy L. Steele's *Common Lisp. The Language*, 2nd ed., 1990, but that hasn't turned out to be a problem.

3.3. The TEX macro `\indexentry`. Spindex uses the conditionals (`\newif`s) `\ifdraft` and `\ifindex` and the control sequences `\suppressindex` and `\firstindexentry`. We've already seen `\ifdraft`; it's used for telling `\indexentry` whether to write a marginal hack or not. The conditional `\ifindex` and the control sequence `\suppressindex` are used for telling TEX whether to make an index or not. The file `spindex.tex` contains the lines

```
\indextrue
%\indexfalse
```

one of which should be commented out, depending on whether you want an index or not. There's another way of suppressing the index, though, without changing `spindex.tex`. The input file can contain the line `\let\suppressindex=t` or `\def\suppressindex{}` before the line `\input spindex`. Then, if `\indextrue`, `\indexfalse` is set instead.

```
\ifindex
\ifx\suppressindex\undefined
\message{\noexpand\indextrue. %
     Will make an index, if there %
     are any entries.}
\else
\indexfalse
\fi\fi
\ifindex
\else
\message{\noexpand\indexfalse. %
     Won't make an index, %
     even if there are entries.}
\fi
```

Then, the definition of `\indexentry` is put inside a conditional using `\ifindex`.

```
\ifindex
\def\indexentry#1#2#3#4#5#6{...}\else
\def\indexentry#1#2#3#4#5#6{\relax}\fi
```

If `\ifindex` expands to `\iffalse` (`\ifindexfalse`), `\indexentry` simply eats its 6 arguments. The control sequences `\firstindexentry` and `\suppressindex` are used as Boolean variables. They can expand to a single token or be undefined, and are used in conditional constructions. Their specific values, if any, are not really important, so I like to use `n` and `t`, like `nil` and `t` in Lisp. The TEX driver file `driver.tex` uses `\suppressindex` the second time TEX is run on it in order to prevent `\indexentry` from overwriting `entries.lsp`.

The line `\let\firstindexentry=t` appears in `spindex.tex`. Assuming `\indextrue`, if the control sequence `\firstindexentry` expands to `t` (i.e., the first time `\indexentry` is invoked), it calls the macro `\beginindex`, which performs certain actions that only need to be performed once. It opens a file called `index_switch.tex` and writes something to it. It doesn't matter what it writes — all `index_switch.tex` has to do is exist. It's used for running Spindex with the UNIX shell script and the TEX driver file described on page 261. TEX cannot directly access shell variables or execute commands in a shell, and a shell script cannot directly influence TEX when it's running. However, both can write and test for the existence of files, so I use `index_switch.tex` to communicate between `run_driver` and `driver.tex`.

We're done with `index_switch.tex` now, so the output stream is closed and freed to be reallocated, if necessary. Now `\beginindex` opens the file which will contain the Lisp code for the index entries. In this article I call it `entries.lsp`, but actually it can have any name within reason. Then it says `\let\firstindexentry=n`, so these actions won't be performed again.

Next, `\indexentry` takes arguments `#2`–`#6` and puts them in boxes. It checks the width of the boxes and behaves appropriately, simulating the effect of true optional arguments. This is a useful trick that does not appear in *The TEXbook*. It's not as neat as a look-ahead mechanism using `\futurelet` or `\afterassignment` and `\let`, but it's a lot easier to code. Here's a simple example of this technique:

```
\setbox2=\hbox{#2}%
\ifdim\wd2>0pt
\message{There's something in %
     argument 2}%
\else
\message{Argument 2 is empty}%
\fi
```

Above I state that six sets of braces must always follow `\indexentry`. Strictly speaking, of course, this isn't true, but TEX will consider the six tokens or groups that follow `\indexentry` to be its arguments, so leaving out the braces (or characters with `\catcode=1` and `\catcode=2`) is hardly practical. The `\indexentry` macro writes code to `entries.lsp` based on what's in its arguments. Argument `#1` is required, so `\indexentry` doesn't need to put it in a box. It writes

(generate-entry @⟨*name*⟩@

The `@` symbol is used as a string delimiter instead of `"` in order to make it possible to use `"` in `\indexentry`'s arguments: `|"a|` for "ä", `|"o|` for "ö", etc. This means that `@` "as is" in an argument to `\indexentry` will cause a fatal error. But `|@|` works. The other arguments are put into boxes.

```
\setbox2=\hbox{#2}%
\setbox3=\hbox{#3}%
\setbox4=\hbox{#4}%
\setbox5=\hbox{#5}%
\setbox6=\hbox{#6}%
```

Then,

```
\ifdim\wd2>0pt
\write\index{\space\space\space %
     :text @#2@}%
\fi
```

causes

:text @⟨*text*⟩@

to be written to `entries.lsp` if `#2` is non-empty, and similarly for the other four arguments, except that `#3` (for suppressing the page number) is treated a little differently, since the page number is printed by default:

```
\ifdim\wd3=0pt
\write\index{\space\space\space %
     :page-no \the\pageno}%
\fi
```

⟹

:page-no ⟨*page number*⟩

if `#3` is empty. After the arguments `#2` through `#6` are tested for existence and the code (if any) is written to `entries.lsp`, `\indexentry` writes a closing parenthesis to match (generate-entry @⟨*name*⟩@. Here are some examples:

```
\indexentry{nouns}{}{}{}{}{}
```

⟹

```
(generate-entry @nouns@
   :page-no 1
   )
```

```
\indexentry{masculine}{masc.}%
     {}{}{nouns}{}
```

⟹

```
(generate-entry @masculine@
   :text @masc.@
   :heading @nouns@
   :page-no 1
   )
```

```
\indexentry{a-stems}{}{x}{verbs}%
     {nouns}{masculine}
```

⟹

```
(generate-entry @a-stems@
   :heading @nouns@
   :subheading @masculine@
   :cross-ref @verbs@
   )
```

```
\indexentry{s|ae|tninger}{}{}%
     {S|"a|tze}{}{}
```

⟹

```
(generate-entry @s|ae|tninger@
   :cross-ref @S|"a|tze@
   :page-no 24
   )
```

The `\write` commands in `\indexentry` are the reason why it can't use the normal coding for macros in its arguments, i.e., the coding using backslashes, like `\th`, `\oe` and `\it`. A `\write` command will expand an expandable macro, and write an unexpandable one as is, but with a following space. There's more about this topic in section 3.6.

After TEX is done with the input file, and all of the index entries have been processed, the output stream `\index` associated with the file `entries.lsp` should be closed. I redefine `\bye` so that it calls the function `\endindex`, which is defined like this:

```
\ifindex
\def\endindex{\closeout\index}
\else
\def\endindex{\relax}
\fi
```

3.4. The Lisp program `spindex.lsp`**.** This program loads the file of Lisp code, `entries.lsp`, which was written by the `\indexentry` commands. This file consists of invocations of the Lisp function generate-entry, which uses `\indexentry`'s `name` argument, and its `heading` and `subheading` arguments, if present, to access a symbol (or variable). Since the names of these symbols depend on the arguments to `\indexentry`, they can be different each time Spindex is run and therefore cannot be declared in `spindex.lsp`. This may appear to be dangerous, but it isn't. Lisp has very few reserved words. Most of its internal variables begin and end in *, like *package*. If an index entry is made with a `name` that duplicates the name of a Lisp function, like car, this will not cause an error (or even a problem), because each Lisp symbol has a function cell and a value as a variable, and the interpreter can tell from the context which is meant. Also, safety routines can be written to catch dangerous names before the string is used to create a symbol. There is one for entries beginning and ending in asterisks, "T" and "NIL". The Gnu Lisp interpreter has named constants that don't begin and end in *, but it will signal an error if an attempt is made to change their values. However, they are represented internally in uppercase letters, and the symbols created by generate-entry probably won't be, so it's unlikely that these constants will cause any problems. If they do, it's still possible to write safety routines to take care of them.

3.4.1. Generating the entries. The name, heading and subheading arguments to generate-entry are all strings and undergo some manipulation before they are used as the names of Lisp symbols. Therefore, some characters may appear in arguments to `\indexentry` which would normally cause problems in Lisp, for instance, an index entry like "Lincoln,␣Abraham" is legal, whereas commas and spaces may not normally appear in symbol names in Lisp. If there is no heading argument, the entry is a heading, and the name of the symbol is name. If heading (but not subheading) is non-empty, the entry is a subheading, and heading and name are joined with a hyphen: heading-name. If heading and subheading are both non-empty, the entry is a subsubheading, and heading, subheading and name are joined with a hyphen, e.g.,

```
\indexentry{transitive}{}{}{}{verbs}{}
```

maps to the symbol name

```
|verbs-transitive|
```

and

```
\indexentry{active}{}{}{}{verbs}%
     {transitive}
```

maps to the symbol name |verbs-transitive-active|.

The use of || surrounding the symbol name in `spindex.lsp` is independent of the use of || to delimit special character codings in `\indexentry`'s arguments. In Lisp, |⟨*characters*⟩ | has the effect of escaping all of the characters inside ||, so that characters can be used in the name of a Lisp symbol that would normally not be allowed. This also makes it possible to have symbol names with lowercase letters. Lisp normally ignores case and converts lowercase letters in symbol names to uppercase letters internally. But this would mean that

```
\indexentry{a}{a (the letter a)}{}{}{}{}
```

and

```
\indexentry{A}{A (the letter A)}{}{}{}{}
```

would map to the same Lisp symbol and therefore not create two different entries, and the `text` "A (the letter A)" would be ignored, because `text` is only used when an entry is created, as explained above. So all lowercase letters are escaped as well as space, comma, and indeed everything except for uppercase letters, which are not escaped, and `{` and `}`, which are ignored.[2] However, this special meaning of | in Lisp means that an index entry for "þat" and one for "that", created by

```
\indexentry{|th|at}{}{}{}{}{}
```

and

```
\indexentry{that}{}{}{}{}{}
```

would both map to a Lisp symbol called |that|, since the || in `|th|at` would be interpreted by Lisp simply as escape characters. In order to prevent this, || in an `\indexentry` are converted to |! and !| so that the two invocations of `\indexentry` above map to two different symbols, |!th!at| and |that|. The exclamation points have no effect on alphabetization or on the output to `index.tex`, since sorting and output both use the original, unconverted name argument.

Now generate-entry accesses the symbol (using read-from-string) and checks to see if it's bound. If it isn't, it means that this is the first occurrence of this entry. In this case, a structure of type "entry" (defined by defstruct entry) with the slots name,

[2] The way characters or groups of characters are handled can be modified according to the user's requirements.

text, sort-string, page-nums, cross-refs, cross-ref-cons, subheadings and subsubheadings is created and the symbol is bound to it. The information in generate-entry's other arguments is stored in the appropriate slots. If the symbol is bound, i.e., the entry already exists, the page number and cross-reference information in generate-entry's arguments may be added to the appropriate slots in the structure, unless it's already there due to previous invocations of `\indexentry`.

It's easier to "fake" an index using the function generate-entry than it is to use a dummy input file. If one wants to type in the code for invocations of generate-entry, there's no need to use `\indexentry` at all, for instance, to make an index for a book that's already been printed or that's not made using TEX. In this case, it would make sense to redefine generate-entry so that it could take lists of strings and integers for its cross-ref and page-num keyword arguments. Then generate-entry need only be invoked once for each entry.

3.4.2. The sort strings. The name argument is used to make a string to be stored in the sort-string slot of the entry structure. This is what makes it possible for Spindex to alphabetize special characters.

Lisp's sorting routine for characters and strings, like C's and UNIX' sorting routines, can sort the 256 characters of an 8-bit character encoding according to a code table based on the ASCII code table. For sorting strings using only English words this is adequate, but most of the special characters likely to appear in an index do not appear in the ASCII code table (or in Lisp's), and most of the characters that *do* appear in the code table are unlikely to appear in an index. Since uppercase letters (positions 65–90) and lowercase letters (positions 97–122) are treated identically for purposes of alphabetization, and it makes no sense to sort numerals or punctuation marks according to their position in the code table, only 26 positions are relevant and 229 are wasted.

Spindex makes it possible to use all 256 positions, or as many of them as necessary, by assigning integer values to a set of variables, i.e., a-value = 1, b-value = 2, etc. Each letter or special character is associated with a list of one or more of these values. The characters a, b and þ are associated with the lists (a-value), (b-value) and (thorn-value) respectively On the other hand, in some languages the ligature "æ" is treated as "ae", so it's associated with the list (a-value e-value). This is the reason for associating characters with lists rather than single integers.[3]

Some characters should be sorted as if they were other characters. All of the uppercase characters should be treated the same as their corresponding lowercase characters, and in some styles of alphabetization "á", "à", "ā", etc. should be treated like "a", so that the list associated with "á" (coded as `\'a` in TEX and `|'a|` in `\indexentry`) should be (a-value). On the other hand, in Icelandic, "á" follows a in the alphabet (likewise for the other vowels), so "á" would need to have a unique value aacute-value such that a-value < aacute-value < b-value. While `spindex.lsp` can assign integer values only from 0 to 255, in practice many more characters can be accommodated, because some characters receive the same values and others use combinations of values assigned to other characters.

The string which was the `name` argument to `\indexentry` is read character by character, except that a `|` causes everything up to the next `|` (a special coding) to be treated as a unit. The function letter-function returns lists of integers to the function generate-info, which creates a new string using the characters from the code table *that have these values*. So, the sort-string for an `\indexentry` "nouns" might look like `"^P^Q^W^P^U"` (consisting of non-printing characters in Lisp's printed representation). It doesn't matter what the sort-string looks like because the user never even needs to know it exists, and the characters which are assigned will vary according to the content of the character list described on page 260. The sort-string for "transitive" might look like

```
"^V^T^A^P^U
 ^V
 ^X^F"
```

where i-value is assigned the integer 10 corresponding to the newline character, as in Fig. 1. The function set-char-values keeps track of how many there are and signals an error if they exceed 256. Spindex can be made to perform alphabetical sorting for languages using non-Latin alphabets if the user makes an appropriate list, or an index can

[3] It would be possible to change the indexing program so that the characters could be associated either with a single integer or a list of integers. If I revise `spindex.lsp` I will probably make this change, but only for aesthetic reasons.

be reversed or scrambled by changing the order of the characters (if anyone wanted to do this).

After the sort string has been generated, it is stored in the entry structure's sort-string slot. Then generate-entry makes a cons cell and puts the sort string into the car and the symbol itself into the cdr.

```
\indexentry{verbs}{}{}{}{}{}
```

$\Longrightarrow$

```
("^X^F^T^B^U" . |verbs|)
```

If the entry is a heading, this cons cell is put into an association list, or alist, called sort-list. If the entry is a subheading, the cons cell is put into an alist in the subheadings slot of the heading entry of which it is a subheading; if it's a subsubheading, it's put into an alist in the subsubheadings slot of the subheading entry of which it is a subsubheading. Got that?[4]

If a subheading is created before its heading exists, e.g.,

```
\indexentry{transitive}{}{}{}{verbs}{}
```

without a preceding

```
\indexentry{verbs}{}{}{}{}{}
```

|verbs| must be created in order for |verbs-transitive| to be stored with its sort string in |verbs|'s subheadings slot. This is accomplished by means of a recursive call to generate-entry. If

```
\indexentry{active}{}{}{}{verbs}%
     {transitive}
```

is invoked before

```
\indexentry{transitive}{}{}{}{verbs}{}
```

|verbs-transitive| is generated by a recursive call to generate-entry, and |verbs|, too, if it doesn't exist already. The page number is suppressed for entries that are generated automatically in this way, and there is no way to specify a `text` for them. This is another reason for putting dummy entries at the beginning of your input file for specifying `text`s.

3.4.3. Page numbers. By default, the macro `\indexentry` writes the page numbers to the file `entries.lsp`. When an entry is created, if the page number has not been suppressed, a list containing the page number is stored in the entry structure's page-nums slot. For each additional call to `\indexentry` the page number (if it hasn't been suppressed) is simply added onto the list, unless that page number is already in the list due to a previous invocation of `\indexentry` on that page. It would be possible to change this in order to keep track of the number of occurrences per page. This is unnecessary for an index, but it might be useful for some other application. Usually, the page numbers will occur in order in the page number list, however, `spindex.lsp` sorts the list before writing the page numbers to `index.tex`, so they will be in the correct order even if the user explicitly changes the page number in the input file with `\pageno=`⟨*integer*⟩ in such a way that the pages are numbered out of order.

3.4.4. Cross-references. A cross-reference (argument `#4` to `\indexentry`) can refer to another entry (at any level) or it can be an arbitrary string. Whichever it is, it is stored as is (the string is not converted) in a list with all the other cross-references for this entry in the cross-refs slot of the entry structure.

When a heading entry is first created, its text argument (or if text is empty, its name argument) is used to make a cons cell that is stored in that entry's cross-ref-cons slot. This is used when this entry is used as a cross-reference in another entry. A subheading entry uses a string consisting of the text or name of its heading, a comma, a space, and its own text or name. A subsubheading entry uses a string consisting of the text or name of its heading, a comma, a space, the text or name of its subheading, a comma, a space, and its own text or name. This string is stored in the cdr of the cons cell, and given to generate-info, which returns a sort-string, which is stored in the car of the cons cell. Cross-references, unlike entries, are always alphabetized according to what is actually printed.

An index entry is illustrated in Fig. 1.

3.4.5. Output. After `spindex.lsp` has loaded the file `entries.lsp`, it puts the cons cells in sort-list (the alist containing the heading entries) into alphabetical order according to their cars, i.e., the sort-strings, with

```
(setq sort-list
   (sort sort-list #'string<
                 :key #'car))
```

Now the heading entries are in alphabetical order and the function export-entries simply pops each cons cell off of sort-list, evaluates the symbol in the cdr to get the entry structure, extracts the information for each entry and writes it to the TEX file `index.tex` (as with `entries.lsp`, any name

[4] The subsubheading slot of a heading entry, the subheading slot of a subheading, and both of these slots in a subsubheading will always be nil.

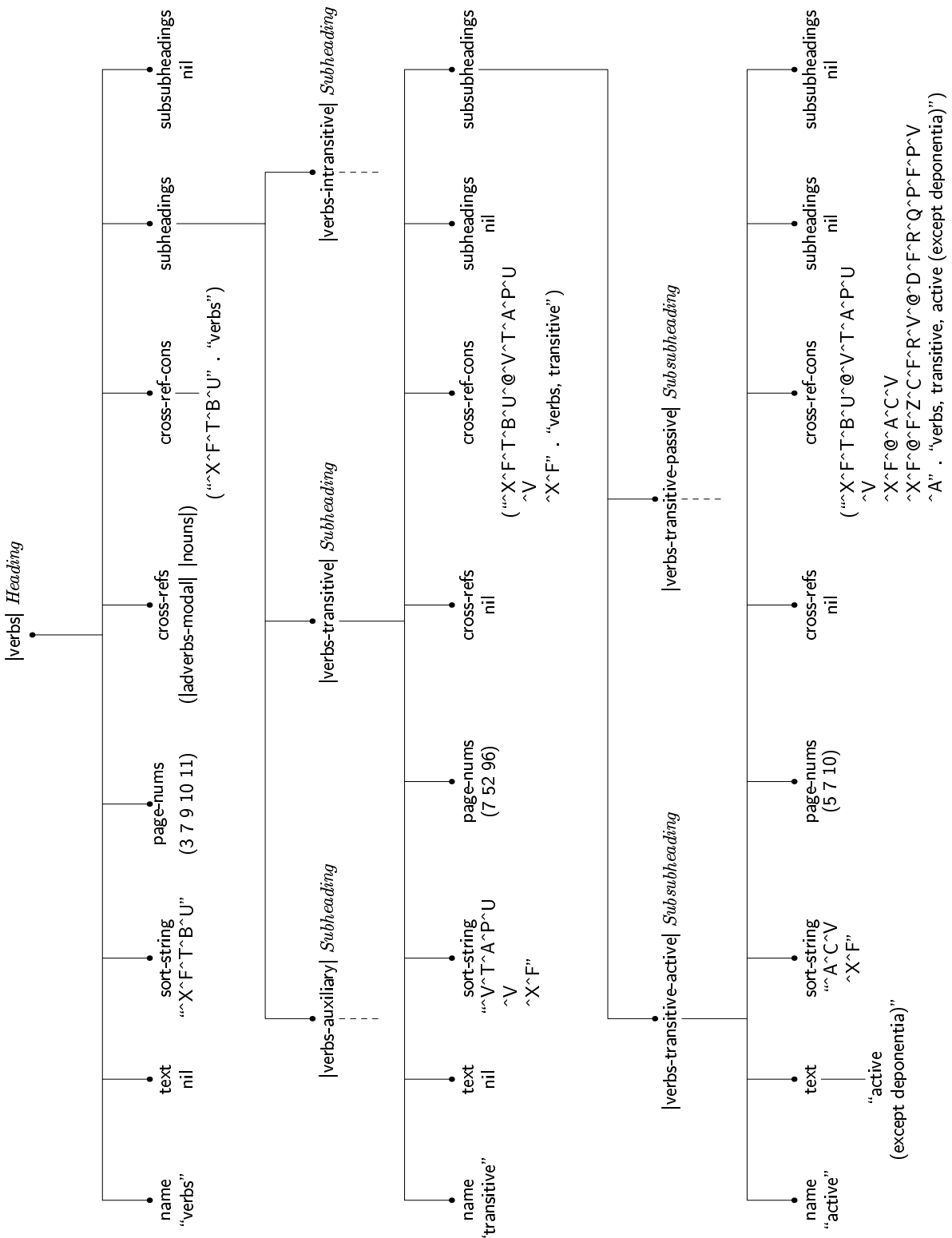


Fig. 1. A heading entry with sub- and subsubheadings.

within reason can be chosen). Headings are not indented, subheadings are indented to the value of `\parindent` and subsubheadings are indented to twice this value. The function generate-info converts the text or name string of each entry into TEX coding, which is written to `index.tex`. When export-entries processes a heading entry, and the subheading slot is non-nil, then the alist in the slot is sorted and export-entries is called recursively. If a subheading entry's subsubheading slot is non-nil, then the alist it contains is sorted and export-entries is called recursively. If there are page numbers associated with an entry, leaders are printed and then the page numbers, separated by commas and followed by a period. It is possible, if unusual, that an `\indexentry` could appear in the front matter, and that the page number would therefore be negative. In this case, export-entries will cause that page number to be printed as a lowercase Roman numeral. If no page numbers are associated with an entry, either because they have all been suppressed, or because an entry was only generated automatically by a sub- or subsubheading entry and `\indexentry` was never called for it in its own right, no leaders are printed. If there are page numbers and cross-references, the cross-references are printed on the following line, indented to the same degree as the entry, preceded by the text "*See also*". If there are cross-references but no page numbers, the cross-references are preceded by the text "*See*". If there are two cross-references, they are separated by the word *and*. If there are more than two, the final two are separated by the word *and* and the others by a semi-colon. Of course, the strings *See*, *See also*, and *and* can be changed for books in languages other than English.

If an entry has no page numbers, no cross-references and there are no sub- or subsubheadings, a warning message is issued. Non-consecutive pages are simply written to `index.tex` and separated by commas. Page ranges are printed as the first and last number in the range, separated by an en-dash (–), whereby the last number may be abbreviated according to the following scheme:

- Let a and b be integers such that $0 < a < b$. a and b represent the beginning and end of a page range.
- If $a < 10^2$, b is not abbreviated: 1–9, 27–100.
- Else if $\lfloor a/10^2 \rfloor = \lfloor b/10^2 \rfloor$ and $(b \bmod 10^2) \geq 10$, b is abbreviated to $(b \bmod 10^2)$: 100–12, 254–99, 1104–29.
- Else if $a \geq 10^3$, $\lfloor a/10^3 \rfloor = \lfloor b/10^3 \rfloor$ and $(b \bmod 10^3) \geq 10^2$, b is abbreviated to $(b \bmod 10^3)$: 1003–125, 2006–194.
- Else if $a \geq 10^4$, $\lfloor a/10^4 \rfloor = \lfloor b/10^4 \rfloor$ and $(b \bmod 10^4) \geq 10^3$, b is abbreviated to $(b \bmod 10^4)$: 10234–1045, 23245–5321.

And similarly for integer $n \geq 5$:

- If $a \geq 10^n$, $\lfloor a/10^n \rfloor = \lfloor b/10^n \rfloor$ and $(b \bmod 10^n) \geq 10^{n-1}$, b is abbreviated to $(b \bmod 10^n)$, up to $b =$ TEX's maximum legal integer (*The TEXbook*, p. 118), namely $2^{31} - 1 = 2147483647 =$ octal $17777777777 =$ hexadecimal 7FFFFFFF: 170234–81045, 1623245–935321, 2037892089–147483647.[5]

Otherwise b is not abbreviated: 102–109, 198–205, 1002–1009, 19052–21088. In particular, page ranges with Roman numerals are never abbreviated: cv–cxii, and page ranges starting with a Roman and ending with an Arabic numeral are impossible. The program in `spindex.lsp` also includes an option for disabling abbreviation.

A possible improvement to Spindex would be to allow page indications followed by *ff*, and underlined and italic page numbers, as in *The TEXbook* and *The METAFONTbook*. This would require changes to `\indexentry` and `spindex.lsp`, but it wouldn't be too difficult. If there is sufficient interest, I will program an option for different styles of page numbering.

If there is more than one cross-reference, they must be sorted alphabetically before they are written to `index.tex`. The same technique is used as for sorting the entries themselves. For an arbitrary string, generate-info generates a sort-string and puts it and the original string into a cons cell. If the cross-reference refers to another entry, the function do-cross-refs gets the cons cell stored in the cross-ref-cons slot of that entry. All of the cons cells are put into a list and sorted according to their cars, i.e., their sort-strings. Then, their cdrs (the original strings) are converted to normal TEX coding by generate-info and written to `index.tex`. If it's an arbitrary string, a warning is issued, that this cross-reference doesn't correspond to an entry.

The formatting of `index.tex` depends on the code written by `spindex.lsp` on the one hand, and

[5] Actually, the Lisp routine that performs the abbreviation can abbreviate integers up to the value of most-positive-long-float using the Gnu Lisp interpreter. On the computer I'm using, it's $1.7977 * 10^{308}$.

on the TEX format used on the other. None of the formatting is hard-wired into the program. The index file can be a complete TEX input file, it can input other TEX files, or it can be input by another TEX file. If the TEX code written to `index.tex` is formulated in a general way, and parameters are set and macros defined in another file, then the same `index.tex` can produce output according to a wide range of different formats without making any changes to the Lisp program. However, it's not difficult to change the TEX code written by export-entries, if the user prefers to do the formatting this way. I do not recommend changing the routines for the page numbers and cross-references, though, unless you know what you're doing.

3.5. Fine points of alphabetization. The function set-char-values assigns values to characters ≥ 1 and < 256. There are, however, two other possible values, nil and 0. If a character is assigned a value of nil, nothing is added to the sort-string and it is ignored for purposes of alphabetization. The value 0 acts as a word separator and is assigned to ␣. This corresponds to one style of alphabetization, namely alphabetization by word, so that an entry "abc xyz" will appear before an entry "abcdef". If nil is assigned to ␣, then the entries will be alphabetized by letter and spaces will be ignored, so "abcdef" will appear before "abc xyz". Other characters, like hyphen, can also act as word separators by assigning them the value 0 (in this case, it's necessary to be careful with em- and en-dashes in arguments to `\indexentry`). Codings using `||` that contain only hyphens and/or spaces (and contain at least one character), are valid and are assigned the value nil, so they can be used when the hyphens and spaces shouldn't act as word separators. The coding `|tie|` is for a ~ that is assigned the value 0 and therefore acts as a word separator. `|tie~nil|` is the coding for a ~ that does not act as a word separator. Characters like $, *, {, }, ?, !, ;, ., :, etc. are assigned the value nil, so they can appear in index entries and do not affect alphabetization. Some codings, like control sequences for font switching or formatting, can also be assigned the value nil, so that the `|it|` in `\indexentry{{|it|abc}}{}{}{}{}{}` does not affect alphabetization. Curly braces in an argument are ignored both for purposes of alphabetization and for accessing symbols, so that `{abc}` and `abc` will map to the same symbol. The coding `|it|abc` will also map to the same symbol as `{|it|abc}`, but the former should not be used because the switch to italic will be global in `index.tex`. Likewise, the user should not type `{|it|␣ abc}` because spaces, even spaces following control sequences, are not ignored for purposes of alphabetization (unless ␣ is assigned the value nil), and `{|it|␣ zzz}` would appear in the index before `{|it|abc}`.

Since some characters are assigned the same values, it's possible for entries that print differently to have identical sort-strings. The two entries

```
\indexentry{a}{a (the letter a)}{}{}{}{}
```

and

```
\indexentry{A}{A (the letter A)}{}{}{}{}
```

will have identical sort-strings, namely `"^A"` (assuming a-value = 1). It is impossible to ensure that lowercase letters will always be sorted before or after uppercase letters in situations like this. The order of these entries in the index will be determined by which of them appeared first in the input file. To ensure a particular order of entries of this type (and to ensure that a `text` argument is not ignored) it is safest to use dummy `\indexentry`s with suppressed page numbers at the beginning of the input file.

Indexes generally do not need to do numerical sorting. If the numerals are all assigned the value nil in letter-function, then entries that differ only with respect to the numerals contained in their names can be put into order by using dummy entries at the beginning of the input file. However, if a particular application requires it, it should be possible to write a routine that will perform true numerical sorting.

3.6. Some limitations. In its current form, Spindex allows three levels of nesting. It is not considered correct form for indexes to have deeper nesting than this, however, it might be desirable for a special purpose, not necessarily for an index. Spindex could be adapted for deeper nesting by adding an argument for each level to `\indexentry`. However, `\indexentry` already has 6 arguments, and it might be desirable to use the remaining three arguments for some other purpose. It is possible to get around TEX's limit of 9 arguments to a macro, but it's easier if one doesn't have to. Macros with lots of arguments encourage typing mistakes and make the input file difficult to read. Modifying `spindex.lsp` would be less of a problem; for each additional level of nesting the entry structures would need an additional slot, and export-entries would need to be called recursively that many more times.

It would be easy to remove the limitation to 256 positions for alphabetical sorting. Let n be an integer such that $n > 0$ and let α be the set of characters processed by set-char-function. Each

character $\in \alpha$ is associated with a single position and assigned a list of n integers. Let β be the set of legal characters $\notin \alpha$ which are assigned lists of n integers, such that each character $\in \beta$ shares a position with a character $\in \alpha$. Let γ be the set of legal characters which are assigned nil. These characters are ignored for purposes of alphabetization, i.e., they are associated with no position. Let δ be the set of legal characters which are associated with lists of integers of length $> n$. The lists assigned to the characters $\in \delta$ may differ in length. For each character $d \in \delta$, let the length of its list be l_d such that l_d is a multiple of n. Then, each character $d \in \delta$ will be associated with x positions such that $x = l_d/n$. Let $\lambda = \alpha \cup \beta \cup \gamma \cup \delta$. Thus λ is the set of legal characters. A string S of length l_S consisting of characters in λ will be associated with y positions where y is the sum of the positions associated with the individual characters in S. Let Z be the sort string derived from S and l_Z its length. Then $l_Z = y * n$. Let p be the number of available positions, then $p = 256^n$. As n increases arithmetically, l_Z increases geometrically and p increases exponentially. If $n = 2$, $p = 256^2 = 65,536$, and for $n = 3$, $p = 256^3 = 16,777,216$. In this way, Spindex can theoretically accommodate infinitely many positions, however, I suspect that increasing n too much would soon cause the Lisp program to run *very slowly* and eventually exhaust the capacity of the computer.

In the format I use, when `\drafttrue`, `\indexentry` causes a marginal hack to be printed next to the line where `\indexentry` appeared in the input file. The marginal hack is printed in the typewriter font `cmtt10`, so an `\indexentry` with `||` like

```
\indexentry{|th|is}{}{}{}{}{}
```

will produce a marginal hack like `|th|is`. If I change the font to roman (`cmr10`), the marginal hack will look like —th—is, because the character — is in same position in `cmr10` as `|` is in `cmtt10` (`"7C`). So I'm limited to using a typewriter font if I want my marginal hacks to look right. Also, two `\indexentry`s on one line will cause the second marginal hack to overwrite the first, causing an unsightly mess. Fixing this would be so complicated that I've decided not to bother, since it's only for rough drafts anyway, and a single line will rarely have multiple invocations of `\indexentry` (except for dummy entries). I'd probably have to define a new class of insertions and I'm not sure it would be possible to get the marginal hacks lined up properly.

Another limitation is that the user can't use normal TEX coding for the special characters and other control sequences in `\indexentry`. Using `||` has advantages, but it would be nice to be able to use normal TEX coding, too.

It is possible to fix this problem, and to have the marginal hack printed in roman type, but the benefit does not justify the increased complexity of `\indexentry`'s definition. However, the solution may be interesting and useful for some other purpose.

To simplify matters, I will use the macro `\next` to illustrate. The following facts are involved:

1. `|` is an ordinary character, `\catcode` = 12.
2. `\write` will expand macros like `\"o`, `\th`, `\it`, the active character `~`, and other active characters like `æ` if such are defined, and put a space after each unexpanded macro, like `\oe`.
3. Changing the `\catcode` of a character used in an argument to a macro has no effect on that character once it's been read and tokenized.
4. `\write` is not executed immediately. It is put into a whatsit and expansion takes place upon `\shipout`. The macros in the text written by `\write` are therefore expanded according to the definitions in force at the time of the `\shipout`, not when `\write` is invoked (*The TEXbook* p. 227).
5. A delayed `\write` must be used (not an `\immediate\write`) in order to write the page number to the opened file.

The problems can be solved in the following way:

```
%%%% This is next.tex

\newwrite\nextout
\immediate\openout\nextout=next.output
\newlinechar=`\^^J

\def\verticalstroke{|}
\def\foo{foo outside}

\catcode`\|=\active
\let|=\verticalstroke

\def\next{\begingroup
\def\foo{foo inside \noexpand\next}
\def|{vertical inside \noexpand\next}
\catcode`\|=\active
\def\subnext##1##2{%
\immediate\write\nextout%
{This is arg1 inside \noexpand\subnext,
^^J but outside the group:^^J##1}
\immediate\write\nextout%
{This is arg2 inside \noexpand\subnext,
^^J but outside the group:^^J##2}
\begingroup
\def\foo{foo inside}%
```

```
26. \def|{vertical inside}%
27. \immediate\write\nextout{This is arg 1
28.      inside \noexpand\subnext,^^J
29.      and inside the group:^^J##1}%
30. \immediate\write\nextout{This is arg 2
31.      inside \noexpand\subnext,^^J
32.      and inside the group:^^J##2}%
33. %%
34. \write\nextout{This is arg 1 at
35.      \noexpand\shipout:^^J
36.      ##1}%
37. \write\nextout{This is arg 2 at
38.      \noexpand\shipout:^^J
39.       ##2}%
40. %% This is for a delayed write of
41. %% the local definitions of the macros
42. %% to \nextout
43. \edef\anext{\write\nextout{^^J%
44.      This is arg 1 at
45.      \noexpand\shipout,^^J
46.      but with the local definition:^^J
47.      ##1}}
48. \anext
49. \edef\anext{\write\nextout{This is arg 2
50.      at \noexpand\shipout,^^J
51.      but with the local definition:^^J
52.      ##2}}%
53. \anext
54. \write\nextout{^^JThis is \noexpand
55.   \catcode\noexpand`\noexpand\|:
56.   \the\catcode`\|}%
57. %% This works
58. \endgroup\endgroup}%
59. \subnext}
60. %% This keeps <macro name> inside \next
61. %% from being written to \nextout
62. %%\endgroup}%
63. %%\expandafter\endgroup\subnext}
64.
65. \catcode`\|=12
66.
67. \next{|}{\foo}
68.
69. \closeout\nextout
70.
71. \end
```

This writes the following text to the file `next.output`

```
This is arg1 inside \subnext ,
 but outside the group:
vertical inside \next
This is arg2 inside \subnext ,
 but outside the group:
foo inside \next
This is arg 1 inside \subnext ,
 and inside the group:
vertical inside
This is arg 2 inside \subnext ,
 and inside the group:
foo inside
This is arg 1 at \shipout :
 |
This is arg 2 at \shipout :
 foo outside

This is arg 1 at \shipout ,
 but with the local definition:
 vertical inside
This is arg 2 at \shipout ,
 but with the local definition:
 foo inside

This is \catcode `\|: 12
```

The `\catcode` of `|` must be set to `\active` outside the definition of `\next`, so that `\def|{...}` will not cause an error. It is set back to 12 (other) after the definition of `\next`. Here, `\subnext` is defined inside of `\next`, but that isn't necessary; it could be defined outside of it, as long as `\catcode`\|=\active` when `\subnext` is defined.

What appear to be arguments to `\next` in line 67 actually are not. Rather, they are arguments to `\subnext`, which therefore must be the last thing in the definition of `\next` before the closing `}`.

Before `\subnext` reads its arguments, `\next` changes the `\catcode` of `|` to `\active`, so it can be defined as a macro. In this example, `|` first expands to `vertical inside \next` and then to `vertical inside` when `\subnext` is expanded. It could also be made to expand to `$\vert$` for a marginal hack, or anything else. At `\shipout`, though, it expands to `|`, i.e., `the character |`. The definition `\def\verticalstroke` in line 7 is necessary to make this possible: because `\catcode`\|=\active`, `\def|{|}` will cause infinite recursion when TEX tries to expand `|`. The definition `\def|{^^7C}` will also fail, because `^^7C` and `|` are equivalent. The `|` in the `\write` command was active when it was tokenized, so it is expanded upon `\shipout` using its global definition, even though `|` is no longer active at this time.

Following this, in lines 40–53, delayed `\write`s are performed using the local definition of `|` and `\foo`. This is accomplished by a trick explained in the answer to Exercise 21.10 of *The TEXbook*:

```
\edef\anext{\write\nextout{##1}}
\anext
```

(a simplified version of the code in line 43–48), causes `|` to be expanded within the definition of `\anext`, before the `\write` command is put into its whatsit. It is, however, necessary to redefine `\anext`

for each argument that is to be written to `\nextout`. Even by taking the definition of `\subnext` out of `\next` (this possibility is mentioned above), which would allow the use of arguments in `\anext`'s definition (arguments to macros whose definitions are as deeply nested as the definition of `\anext` is here are not possible, since TEX does not allow parameters like `###1`), and writing

```
\edef\anext##1{{\write\nextout{##1}}%
\anext#1
\anext#2
\anext#3
```

won't work — `vertical outside` and `foo outside` will be written to `\nextout`, apparently because the local definitions of `|` and `\foo` are not accessible inside of `\anext`, but I really don't know the reason.

Macros need not be redefined before the arguments are read. By using grouping, it's possible to have `\subnext` expand the macros in three different ways (or as many as TEX's memory allows), depending on the time of expansion, as in the example above. However, if delayed `\write` commands are used, and the token lists are not expanded beforehand using an `\edef`, it is important to make sure that all macros in the text to be written are defined at the time of `\shipout`. If a macro is only defined within a group, and the group has ended when `\shipout` occurs, it will cause an "undefined control sequence" error.

The group begun in `\next` ends at the end of `\subnext`. If `\endgroup` was placed after `\subnext` is called at the end of `\next`, it would be interpreted as `\subnext`'s first argument. It also doesn't work to write `\expandafter\endgroup\subnext` in line 59 (and remove one of the `\endgroups` in line 58). This will have the effect that `vertical inside \next` and `foo inside \next` are never printed to `next.output`, since these definitions will be inaccessible to `\subnext`. I admit, I don't know why this is. It seems that TEX temporarily "forgets" it's in this group while it's expanding `\subnext`.

4. Final remarks

Spindex runs TEX on an input file which writes information to a file of Lisp code. A Lisp program inputs this file and writes another TEX file. This is only one possibility of using TEX and an auxiliary program in combination. Spindex needs to run TEX initially in order to generate page number information by means of TEX's output routine. This may not be necessary for other applications, so another auxiliary program might operate directly on the TEX input file. Another possibility is storing data in files of Lisp code and using a Lisp program to generate TEX input files. Of course, auxiliary programs can be written in other languages, like C, Fortran, Pascal, etc.

Auxiliary programs like Spindex depend on the fact that TEX input files are ASCII files. The value of this feature of TEX doesn't seem to be recognized as much as it ought to be. It would be impossible, or at the very least impractical, for an amateur (like me) to implement an indexing program for a word-processing package that stores its typesetting data in a format that people can't read. The trend in software is clearly in favor of menu-driven, point-and-shoot programs with colorful graphics and sound effects. While programs of this sort are superficially easier to use than packages like TEX and METAFONT, they discourage creativity on the part of the user, at least with respect to programming extensions to the programs themselves.

LATEX presents a similar problem. The more macros you use, the more likely it is that a macro you write will cause an unforeseen problem, especially if you don't understand how the macros you're using work. Large packages offer functionality, which is not always needed, and you pay for it with increased run-time and a loss of flexibility. I used LATEX when I first started writing auxiliary programs, but I found that I spent most of my time trying to make it stop doing things that I didn't want. For this reason (among others), I recommend using `plain` TEX, and the other formats and macros documented in *The TEXbook*, as the basis for programming extensions to TEX.

I've used some of the other possible combinations of TEX and auxiliary programs in other packages, which I plan to document in subsequent articles. Many of the techniques described in this article are of general applicability, not just for indexing. I hope that Spindex may inspire other TEX users to try writing an auxiliary program of their own.

◇ Laurence Finston
Skandinavisches Seminar
Georg-August-Universität
Humboldtallee 13
D-37073 Göttingen
Germany
lfinsto1@gwdg.de

Graphics Applications

Creating 3D animations with METAPOST

Denis Roegel

Abstract

METAPOST can be used to create animations. We show here an example of animation of polyhedra, introducing the 3d package.

Introduction

METAPOST (Hobby (1992); see also the description in Goossens et al. (1997)) is a drawing language very similar to METAFONT, but whose output is PostScript. METAPOST is especially suited for geometrical and technical drawings, where a drawing can naturally be decomposed in several parts, related in some logical way. Knuth is using METAPOST for the revisions of and additions to *The Art Of Computer Programming* (Knuth, 1997), and it is or will be a component of every standard TEX distribution.

Unfortunately, METAPOST is still quite bare and the user is only offered the raw power — a little bit like the TEX user who only has plain TEX at his/her disposal. The lack of libraries is certainly due to the infancy of METAPOST (which came in the public domain at the beginning of 1995) and thus to the small number of its users.

In this paper, we present a way to produce animations using METAPOST. The technique is quite general and we illustrate it through the 3d package.

Animations

The World Wide Web has accustomed us to various animations, especially java animations. Common components of web pages are animated GIF images.

Producing animations in METAPOST is actually quite easy. A number of n images will be computed and their sequence produces the animation. The animation will be similar to a movie, with no interaction. More precisely, if an_image(i) produces a picture parameterized by i, it suffices to wrap this macro between beginfig and endfig:

```
def one_image_out(expr i)=
  beginfig(<figure number>);
    an_image(i);
  endfig;
enddef;
```

and to loop over one_image_out:

```
for j:=1 upto 100:one_image_out(j);endfor;
```

Assuming that <figure number> is equal to the parameter of an_image, the compilation of this program will produce a hundred files with extensions .1, .2, . . . , .100. All these files are PostScript files and all we need to do is to find a way to collate them in one piece. How to do this depends on the operating system. On UNIX for instance, one can use Ghostscript to transform a PostScript file into ppm and then transform each ppm file into GIF with ppmtogif. These programs are part of the NETPBM package (Davidsen, 1993). Finally, a program such as gifmerge (Müller, 1996) creates an animated GIF file (GIF89A) out of the hundred individual simple GIFs. However, various details must be taken care of. For instance, only a part of Ghostscript's output is needed and selection can be made with pnmcut.

The whole process of creating an animation out of METAPOST's outputs can be summed up in a shell script, similar to the one in figure 1. As we will see, this script (including the arguments of awk and pnmcut) can be generated automatically by METAPOST itself.

Objects in space

Introduction The author applied this idea to the animation of objects in space. The macros in the 3d.mp package[1] provide a basis for the representation of three-dimensional objects. The basic components of the objects are the points or the vectors. Both are stored as triplets. More precisely, we have three arrays[2] of type numeric:

```
numeric vect[]x,vect[]y,vect[]z;
```

Vector i's components are vect[i]x, vect[i]y and vect[i]z. It is then straightforward to define the usual operations on vectors using this convention. For instance, vector addition is defined as:

```
def vect_sum(expr k,i,j)=
  vect[k]x:=vect[i]x+vect[j]x;
  vect[k]y:=vect[i]y+vect[j]y;
  vect[k]z:=vect[i]z+vect[j]z;
enddef;
```

[1] On CTAN, under graphics/metapost/macros/3d. The code is documented with MFT (Knuth, 1989) and illustrated with METAPOST. This paper describes version 1.0 of the macros.

[2] METAPOST has a few simple types such as numeric, boolean, string, path, ... It also has pairs (pair) and triples (color). We might have cheated and stored points as colors, but instead, we found it interesting to illustrate a construction equivalent to PASCAL's records or C's structures. In METAPOST, instead of having a list or an array of structures, we use several lists or arrays, so that a record is a cross-section over several arrays.

```
#! /bin/sh

/bin/rm -f animpoly.log
for i in `ls animpoly.*| grep 'animpoly.[0-9]'`;do
echo $i
echo '=============='
# shift each picture so that it lies in the page:
awk  < $i '{print} /^%%Page: /{print "172 153 translate\n"}' > $i.ps
# produce ppm format:
gs -sDEVICE=ppmraw -sPAPERSIZE=a4 -dNOPAUSE -r36 -sOutputFile=$i.ppm -q -- $i.ps
/bin/rm -f $i.ps
# produce gif:
ppmquant 32 $i.ppm | pnmcut 15 99 141 307 | ppmtogif > `expr $i.ppm : '\(.*\)ppm'`gif
/bin/rm -f $i.ppm
done
/bin/rm -f animpoly.gif
# merge the gif files:
gifmerge -10 -l1000 animpoly.*.gif > animpoly.gif
/bin/rm -f animpoly.*.gif
```

Figure 1: Script created by METAPOST (with some additional comments)

Often, we need some scratch vectors or vectors local to a macro. A simple vector allocation mechanism solves the problem: we use a stack of vectors and we reserve and free vectors only on top of the stack. For instance, the allocation of a vector is defined by:

```
def new_vect=incr(last_vect_) enddef;
```

where `last_vect_` is the index of the top of the stack. Hence, a vector is manipulated by its index on the stack. Writing `v:=new_vect;` lets `v` be the index of the newly allocated vector.

Freeing a vector is also easy and is only allowed at the top of the stack:

```
def free_vect(expr i)=
  if i=last_vect_:
     last_vect_:=last_vect_-1;
  else: errmessage("Vector " &
     decimal i & " can't be freed!");
  fi;
enddef;
```

How these macros are used is made explicit in the `vect_rotate` macro which does a rotation of a vector `v` around a vector `axis` by an angle `alpha`. This rotation is illustrated in figure 2. $\vec{v}$ is written as the sum of $\vec{h}$ and $\vec{a}$ where $\vec{h} \perp \vec{a}$. If $\vec{b}$ is $\overrightarrow{axis}/\|\overrightarrow{axis}\|$, $\vec{c}$ is computed as the vector product of $\vec{b}$ and $\vec{a}$ and $\vec{a}$ is then rotated in a simple way resulting in $\vec{f}$.

The vectors declared with `new_vect` are freed in the inverse order. The `vect_rotate` macro makes use of a few other macros: `vect_mod` computes the modulus of a vector; `vect_dprod(a,b)` is the dot product of vectors `a` and `b`; `vect_mult(b,a,x)` lets vector `b` equal vector `a` multiplied by the scalar `x`; `vect_sum` and `vect_diff` compute as their first argument the sum or the difference of the two other vectors; `vect_prod(c,a,b)` lets vector `c` equal the vectorial product of vectors `a` and `b`. These macros are described in appendix A.

```
vardef vect_rotate(expr v,axis,alpha)=
  save v_a,v_b,v_c,v_d,v_e,v_f;
  v_a:=new_vect;v_b:=new_vect;
  v_c:=new_vect;v_d:=new_vect;
  v_e:=new_vect;v_f:=new_vect;
  v_g:=new_vect;v_h:=new_vect;
  vect_mult(v_b,axis,1/vect_mod(axis));
  vect_mult(v_h,v_b,vect_dprod(v_b,v));
  vect_diff(v_a,v,v_h);
  vect_prod(v_c,v_b,v_a);
  vect_mult(v_d,v_a,cosd(alpha));
  vect_mult(v_e,v_c,sind(alpha));
  vect_sum(v_f,v_d,v_e);
  vect_sum(v,v_f,v_h);
  free_vect(v_h);free_vect(v_g);
  free_vect(v_f);free_vect(v_e);
  free_vect(v_d);free_vect(v_c);
  free_vect(v_b);free_vect(v_a);
enddef;
```

The `3d` package defines other macros in order to set the observer, to compute a reference matrix, etc. Provision is given for manipulating objects.

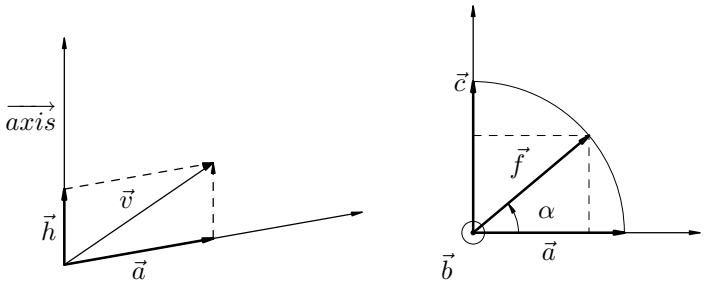


Figure 2: Vector rotation

Objects and classes The `3d` package understands a notion of *class*. A *class* is a parameterized object. For instance, we have the class of regular tetrahedra, the class of regular cubes, etc. Our classes are the lowest level of abstraction and classes can not be composed. They can only be *instanciated*. When we need a specific tetrahedron, we call a generic function to create a tetrahedron, but with an identifier specific to one instance.

A class is a set of vertices in space, together with a way to draw faces, and therefore edges. The author's focus was to manipulate (and later animate) polyhedra. As an example, the `poly.mp` package provides the definition of each of the five regular convex polyhedra.

Each class consists of two macros: one defines the points, the other calls the first macro and defines the faces. Each macro has a parameter which is a string identifying the particular instance of that class.

The points of a regular tetrahedron are defined in `set_tetrahedron_points`, an example of the general macro name `set_`⟨*class*⟩`_points`. Five points are defined, four of them with `set_obj_point`, a macro which defines points *local* to an object. The first four points are the vertices and the fifth is the center of the tetrahedron. `set_obj_point`'s first parameter is the point number and the other three are the cartesian coordinates. The first three points are in a plane and the fourth is obtained with the `new_face_point` macro, which folds a face (see the description in appendix A for more details). The `new_face_point` macro is used with the angle `an` which is computed in advance. Once the four points are set, the object is normalized, which means that it is centered with respect to the list of vertices given as parameter (here `1,2,3,4`) and the last vertex is put on a sphere of radius 1, centered on the origin. Therefore, point 5 is the center of the tetrahedron, and the tetrahedron is set symmetrically with respect to the origin.

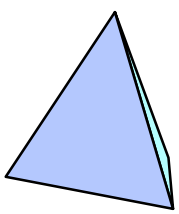
All five convex regular polyhedra are defined in this way and may therefore be inscribed in a sphere of radius 1.

```
def set_tetrahedron_points(expr inst)=
  set_obj_point(1,0,0,0);
  set_obj_point(2,1,0,0);
  set_obj_point(3,cosd(60),sind(60),0);
  sinan=1/sqrt(3);
  cosan=sqrt(1-sinan**2);
  an=180-2*angle((cosan,sinan));
  new_face_point(4,1,2,3,an);
  normalize_obj(inst)(1,2,3,4);
  set_obj_point(5,0,0,0);
enddef;
```

The second macro, `def_tetrahedron` defines the number of points and faces of the instance, calls the previous macro and defines the faces with the macro `set_obj_face`. The first argument of that macro is a *local* face number, the second is a list of vertices such that the list goes clockwise when the face is visible. The last argument is the color of the face in RGB.

```
vardef def_tetrahedron(expr inst)=
  new_obj_points(inst,5);
  new_obj_faces(inst,4);
  set_tetrahedron_points(inst);
  set_obj_face(1,"1,2,4","b4fefe");
  set_obj_face(2,"2,3,4","b49bc0");
  set_obj_face(3,"1,4,3","b4c8fe");
  set_obj_face(4,"1,3,2","b4fe40");
enddef;
```

The result of the drawing is:



A more complex example is the icosahedron which is defined below.

```
def set_icosahedron_points(expr inst)=
  set_obj_point(1,0,0,0);
  set_obj_point(2,1,0,0);
  set_obj_point(3,cosd(60),sind(60),0);
  cosan=1-8/3*cosd(36)*cosd(36);
  sinan=sqrt(1-cosan*cosan);
  an=180-angle((cosan,sinan));
  new_face_point(4,1,2,3,an);
  new_face_point(5,2,3,1,an);
  new_face_point(6,3,1,2,an);
  new_face_point(7,2,4,3,an);
  new_face_point(8,3,5,1,an);
  new_face_point(9,1,6,2,an);
  new_face_point(10,3,4,7,an);
  new_face_point(11,3,7,5,an);
  new_face_point(12,1,8,6,an);
  % 1 and 10 are opposite vertices
  normalize_obj(inst)(1,10);
  % center of icosahedron
  set_obj_point(13,0,0,0);
enddef;
```

```
vardef def_icosahedron(expr inst)=
  save cosan,sinan,an;
  new_obj_points(inst,13);
  new_obj_faces(inst,20);
  set_icosahedron_points(inst);
  set_obj_face(1,"3,2,1","b40000");
  set_obj_face(2,"2,3,4","ff0fa1");
  set_obj_face(3,"3,7,4","b49b49");
  set_obj_face(4,"3,5,7","b49bc0");
  set_obj_face(5,"3,1,5","b4c8fe");
  set_obj_face(6,"1,8,5","b4fefe");
  set_obj_face(7,"1,6,8","b4fe40");
  set_obj_face(8,"1,2,6","45d040");
  set_obj_face(9,"2,9,6","45a114");
  set_obj_face(10,"2,4,9","45a1d4");
  set_obj_face(11,"9,4,10","4569d4");
  set_obj_face(12,"4,7,10","112da1");
  set_obj_face(13,"7,5,11","b4fefe");
  set_obj_face(14,"5,8,11","b49bc0");
  set_obj_face(15,"8,6,12","45a114");
  set_obj_face(16,"6,9,12","b49b49");
  set_obj_face(17,"8,12,11","b40000");
  set_obj_face(18,"7,11,10","45a1d4");
  set_obj_face(19,"12,10,11","b4c8fe");
  set_obj_face(20,"9,10,12","ff0fa1");
enddef;
```

Since all points of the objects are stored in a unique global array, they are internally accessed by the local numbers and an offset defined by the macro `new_obj_points`. The icosahedron example shows a systematic use of the `new_face_point` macro to compute a point on an adjacent face. Displaying such an icosahedron results in the figure 3.

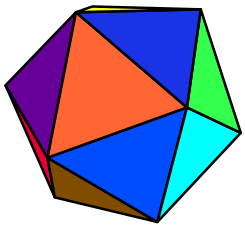
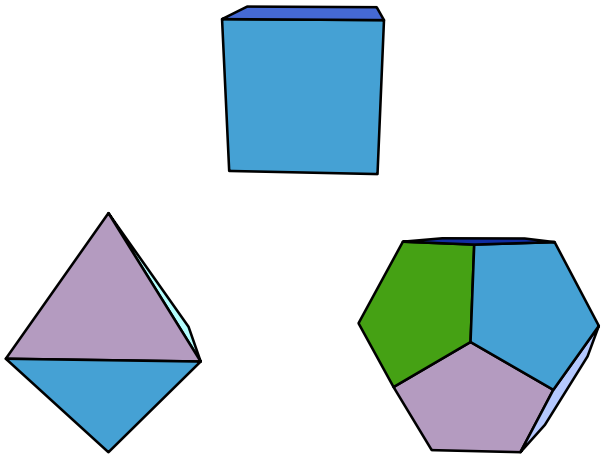


Figure 3: An icosahedron

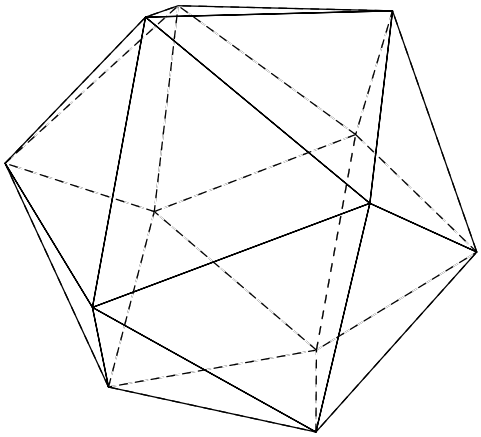
The other three regular convex polyhedra are:



The dodecahedron code is a bit special, since the vertices are built using ten additional points corresponding to face centers. These points are defined as an array of variables `fc1` through `fc10` with `new_points(fc)(10)`. `free_points(fc)(10)` frees them when they are no longer necessary. An excerpt of the dodecahedron code is:

```
def set_dodecahedron_points(expr inst)=
  new_points(fc)(10);% face centers
  set_point(fc1,0,0,0);
  set_obj_point(1,1,0,0);
  set_obj_point(2,cosd(72),sind(72),0);
  rotate_in_plane(3,fc1,1,2);
  ...
  free_points(fc)(10);
enddef;
```

Finally, wire drawings can be obtained by setting the boolean `filled_faces` to false:



Animating objects The animation of one or several objects involves the object(s) and an observer. The animation is a set of images and from an image to the next one, the observer as well as the objects can move. For instance the macro `one_image` in `3d.mp` is:

```
def one_image(expr name,i,a,rd,ang)=
  beginfig(i);
    set_point(Obs,
     -rd*cosd(a*ang),-rd*sind(a*ang),1);
    Obs_phi:=90;Obs_dist:=2;
    % fix point 1 of object |name|
    point_of_view_obj(name,1,Obs_phi);
    draw_obj(name);
    rotate_obj_pv(name,1,vect_I,ang);
    % show the rotation point
    draw_point(name,1);
    draw_axes(red,green,blue);
  endfig;
enddef;
```

The parameters of this macro are a name of an object (`name`), an image index (`i`), and three values defining the position of the observer. The observer (`Obs` is a global point and set with `set_point`, not with `set_obj_point`) follows a circle of radius `rd`. The parameter `a`, which is usually a function of `i`, determines the number of rotation steps of the observer, each step being a rotation of angle `ang`. The distance between the observer and the projection plane is 2 (see figure 4).

The orientation of the observer is defined by three angles (see figure 5). The `Obs_phi` angle is given and the two others are computed with a call to `point_of_view_obj(name,1,Obs_phi)` which constrains the observer to look towards point 1 of object `name`. Therefore, this point will seem fixed on the animation and `draw_point(name,1)` draws it later so that this feature can be observed. There is nothing special about that point, except that it remains fix when the object is rotated. The object is drawn with `draw_obj(name)` and `rotate_obj_pv` rotates the object `name` by `ang` degrees around an axis going through point 1 and directed by vector `vect_I` ($\vec{\imath}$). The reference vectors ($\vec{\imath}$, $\vec{\jmath}$ and $\vec{k}$) are drawn in red, green and blue with `draw_axes`.

Finally, a complete animation of an icosahedron is obtained with

```
animate_object("icosahedron",1,100,100);
```

which generates files `anim.101`, ..., `anim.200` from the main file `anim.mp`. The first parameter of `animate_object` is the name of the object to animate, the second and third parameters are minimal and maximal values of the index loop and the fourth parameter is an offset added to the index loop in order to get the file extension, which must lie in the interval 0..4096.

After each image is drawn, the values of the current bounding box are used to compute the bounding box of the sequence of images. The internal values `xmin_`, `ymin_`, `xmax_` and `ymax_` hold the minimal and maximal values of the coordinates of the past images' corners. They are updated just before each image is shipped out.

Putting the pieces together Once all the views have been computed, they can be used separately (see for instance the five views of figure 6) or more interestingly, they can be merged. This task is made almost straightforward by METAPOST itself. Indeed, every time `animate_object` is used, a shell script named `create_animation.sh` is generated, as a side-effect of a call to `show_animation_bbox`. The script is similar to that shown in figure 1. This script uses the values computed for the global bounding box of the sequence of images, for these values are necessary in order to extract the right parts of the images and get correct alignments; the parts are extracted with `pnmcut`.[3] If you have the programs used in this script (`Ghostscript`, etc.), you can just run it with `sh create_animation.sh` on UNIX. You may need to adapt it to your needs, and for that purpose, you can modify the macro `write_script` in `3d.mp`.

Some examples are included in the `3d` distribution, and they can be viewed for instance with `netscape` or special programs such as `xanim`.

[3] One might think of using `Ghostscript` for generating an excerpt of an image, but if `Ghostscript` is used to generate the bounding box of an image, it will in general not be possible to have a good alignment between all images. The sizes of the excerpts are only known when all images have been produced.

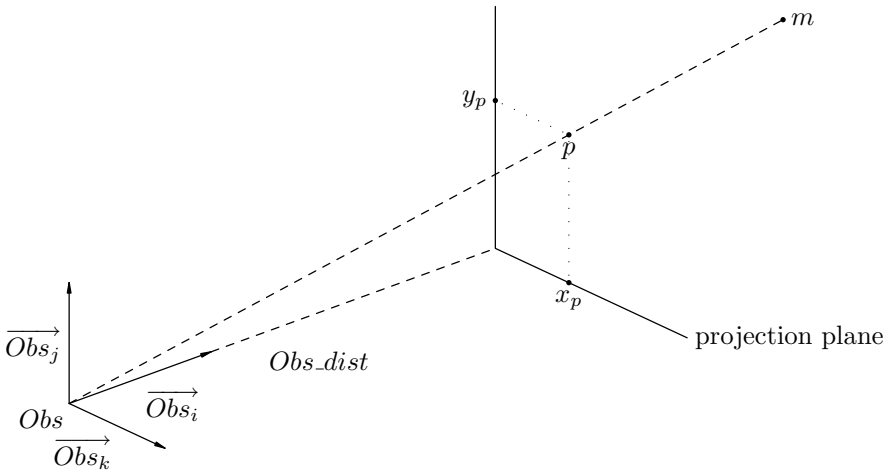


Figure 4: Projection on the screen

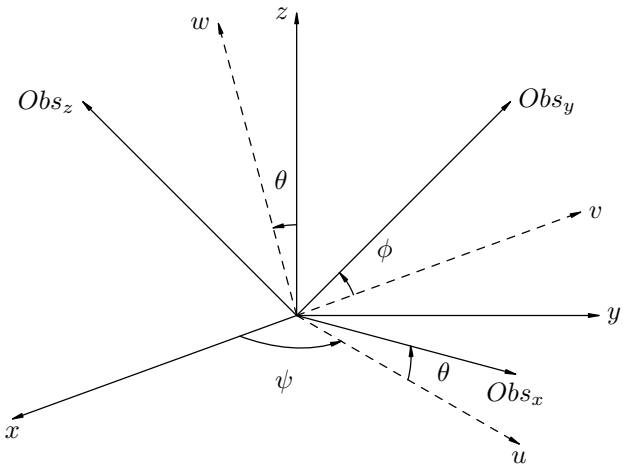


Figure 5: Orientation of the observer

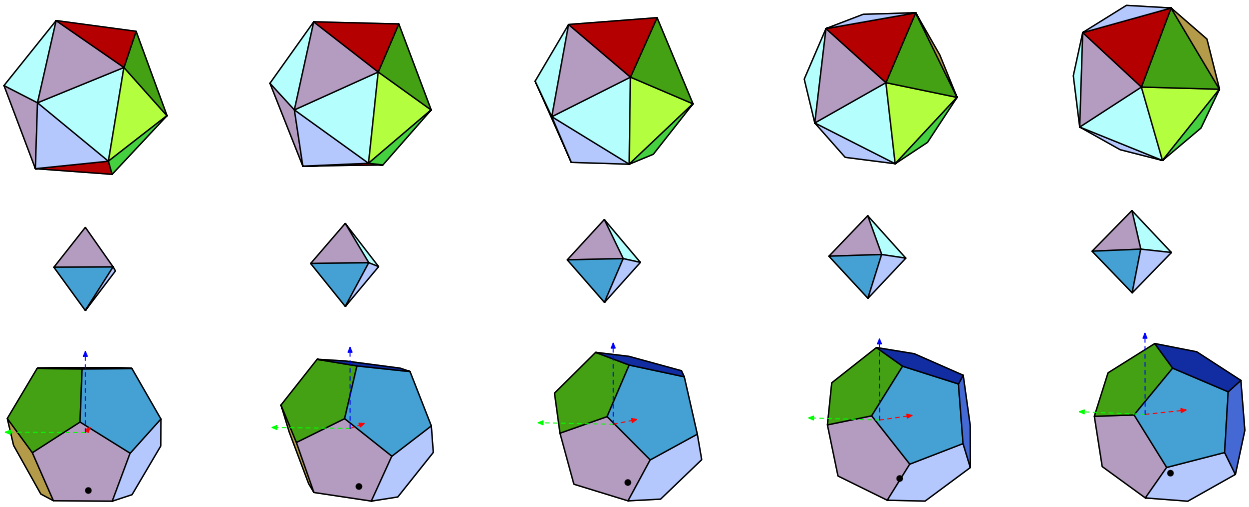


Figure 6: Five views of an animation

Future

It is quite easy to improve and extend the `3d` macros but the author decided to go no further for the moment. Other objects can be implemented easily and new algorithms can be added. For instance in order to take light sources or shadows into account, one can compute the angles under which a face gets its light, and the angle under which this very face is seen, in order to decide how much darker or lighter it must be rendered. Another problem is to represent overlapping objects correctly. In the current implementation, each object is drawn independently from the other objects, so that the overlapping may be wrong. One solution is to sort all the faces according to their distance to the observer and, if two faces can not be ordered, to split them. Then, the faces can be drawn starting with the most distant, and ending with the closest one. Appendix B explains the internal representation of the objects and shows that this algorithm can be implemented without much surgery to the present code.

Acknowledgments

Thanks to John Hobby who always answers all my queries on the METAFONT mailing list. Thanks to Alain Filbois who helped me with the shell script syntax, to Thomas Lambolais and Thomas Genet who gave some feedback on this work, and to Dominique Larchey who pointed out a shortcoming in the conclusion. Thanks to Denis Barbier who was one of the first users of these macros and contributed the animated crayons in the distribution. Thanks to Bogusław Jackowski who made valuable comments on some peculiarities of the code. And finally, special thanks to Ulrik Vieth who not only pushed me to polish my code and this paper more than I had first intended, but also made it possible to use METAPOST under `web2c`.

References

Davidsen, Bill. "NETPBM". 1993. Available for instance at `ftp://ftp.wustl.edu/graphics/graphics/packages/NetPBM`.

Goossens, Michel, S. Rahtz, and F. Mittelbach. *The LaTeX Graphics Companion*. Addison-Wesley, Reading, MA, USA, 1997.

Hobby, John D. "A User's Manual for MetaPost". Technical Report 162, AT&T Bell Laboratories, Murray Hill, New Jersey, 1992.

Knuth, Donald E. "MFT, version 2.0". 1989. Standard TeX distribution.

Knuth, Donald E. *The Art of Computer Programming, volumes 1, 2 and 3*. Addison-Wesley Publishing Company, 1997. New editions.

Müller, René K. "GIFMerge, version 1.33". 1996. Available at `http://www.iis.ee.ethz.ch/~kiwi/GIFMerge/`.

Appendix A
Summary of the `3d` package

Types The commands in the `3d` package take parameters of several different types. The types are described here.

- An ⟨*avn*⟩ (*Absolute Vector Number*) is the internal number identifying a vector in the `vect` array (an integer).
- An ⟨*apn*⟩ (*Absolute Point Number*) refers to a vector in the same way as an ⟨*avn*⟩ (an integer).
- A ⟨*lpn*⟩ (*Local Point Number*) is a number identifying a point *within* an object (an integer). Two ⟨*lpn*⟩s with the same value can correspond to different points in different objects.
- An ⟨*afn*⟩ (*Absolute Face Number*) is the internal number identifying a face.
- A ⟨*lfn*⟩ (*Local Face Number*) is a number identifying a face *within* an object (an integer). As for points, two ⟨*lfn*⟩s with the same value can correspond to different faces in different objects.
- A ⟨*cl*⟩ (*Class*) is a string representing a class, for instance `"tetrahedron"`. It may only contain letters and underscores.
- An ⟨*obj*⟩ (*Object*) is a string representing an object, that is an instance of a class. Such a string may only contain letters and underscores.
- A ⟨*vl*⟩ (*Vertex List*) is a list of integers, where each integer identifies a vertex. For instance, `1,7` is the list of vertices 1 and 7.
- A ⟨*vsl*⟩ (*Vertex String List*) is a string corresponding to a list of integers, where each integer identifies a vertex. For instance, `"1,2,6,5"` is the list of vertices 1, 2, 6 and 5.
- ⟨*hc*⟩ (*Hex Color*) is a string representing a color with the three RGB components in hexadecimal and in the range 0..255. For instance, `"b4fe40"`.
- ⟨*col*⟩ (*Color*) is a standard METAPOST color (a triplet of RGB components in the range 0..1), such as `red`.
- ⟨*str*⟩ (*String*) is a string.
- ⟨*pair*⟩ (*Pair*) is a pair of numerics.
- ⟨*num*⟩ (*Numeric*) is a number.
- ⟨*bool*⟩ (*Boolean*) is a boolean.

Low level vector commands The low level vector commands define the classical operations in vector algebra.

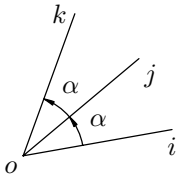
- `vect_def(`$\langle avn\rangle$`,`x`,`y`,`z`)`: defines vector $\langle avn\rangle$ as (x, y, z);
- `set_point`; synonym of `vect_def`: a point is stored in the same array as vectors.
- `set_obj_point(`$\langle lpn\rangle$`,`x`,`y`,`z`)`: this defines the point $\langle lpn\rangle$ as (x, y, z);
- `vect_def_vect(`$\langle avn\rangle_1$`,`$\langle avn\rangle_2$`)`: vector $\langle avn\rangle_1$ becomes equal to vector $\langle avn\rangle_2$;
- `vect_sum(`$\langle avn\rangle_1$`,`$\langle avn\rangle_2$`,`$\langle avn\rangle_3$`)`: the vector $\langle avn\rangle_1$ becomes the sum of vectors $\langle avn\rangle_2$ and $\langle avn\rangle_3$.
- `vect_translate(`$\langle avn\rangle_1$`,`$\langle avn\rangle_2$`)`: add vector $\langle avn\rangle_2$ to vector $\langle avn\rangle_1$; vector $\langle avn\rangle_2$ remains unchanged.
- `vect_diff(`$\langle avn\rangle_1$`,`$\langle avn\rangle_2$`,`$\langle avn\rangle_3$`)`: the vector $\langle avn\rangle_1$ becomes the difference between vectors $\langle avn\rangle_2$ and $\langle avn\rangle_3$.
- `vect_dprod(`$\langle avn\rangle_1$`,`$\langle avn\rangle_2$`)` $\rightarrow \langle num\rangle$: returns the dot product of vectors $\langle avn\rangle_1$ and $\langle avn\rangle_2$.
- `vect_mod(`$\langle avn\rangle$`)` $\rightarrow \langle num\rangle$: returns the modulus of vector $\langle avn\rangle$.
- `vect_prod(`$\langle avn\rangle_1$`,`$\langle avn\rangle_2$`,`$\langle avn\rangle_3$`)`: the vector $\langle avn\rangle_1$ becomes the vector product of vectors $\langle avn\rangle_2$ and $\langle avn\rangle_3$.
- `vect_mult(`$\langle avn\rangle_1$`,`$\langle avn\rangle_2$`,`$\langle num\rangle$`)`: $\langle avn\rangle_1$ becomes vector $\langle avn\rangle_2$ scaled by $\langle num\rangle$.
- `mid_point(`$\langle avn\rangle_1$`,`$\langle avn\rangle_2$`,`$\langle avn\rangle_3$`)`: vector (or point) $\langle avn\rangle_1$ becomes the mid-point of vectors (or of the line joining the points) $\langle avn\rangle_2$ and $\langle avn\rangle_3$.
- `vect_rotate(`$\langle avn\rangle_1$`,`$\langle avn\rangle_2$`,`a`)`: vector $\langle avn\rangle_1$ is rotated around vector $\langle avn\rangle_2$ by the angle a.

Operations on objects Several operations apply globally on objects:

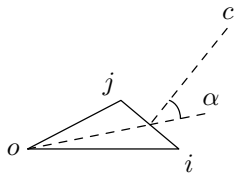
- `assign_obj(`$\langle obj\rangle$`,`$\langle cl\rangle$`)`: create $\langle obj\rangle$ as an instance of class $\langle cl\rangle$.
- `reset_obj(`$\langle obj\rangle$`)`: put $\langle obj\rangle$ back where it was just after it was initialized.
- `put_obj(`$\langle obj\rangle$`,`$\langle avn\rangle$`,`s`,`ψ`,`θ`,`ϕ`)`: object $\langle obj\rangle$ is scaled by s, shifted by vector $\langle avn\rangle$ and oriented with the angles ψ, θ, ϕ, as for the observer orientation (figure 5).
- `rotate_obj_pv(`$\langle obj\rangle$`,`$\langle lpn\rangle$`,`$\langle avn\rangle$`,`a`)`: object $\langle obj\rangle$ is rotated around an axis going through local point $\langle lpn\rangle$ and directed by vector $\langle avn\rangle$; the rotation is by a degrees.
- `rotate_obj_abs_pv(`$\langle obj\rangle$`,`$\langle apn\rangle$`,`$\langle avn\rangle$`,`a`)`: the object $\langle obj\rangle$ is rotated around an axis going through absolute point $\langle apn\rangle$ and directed by vector $\langle avn\rangle$; the rotation is by a degrees.
- `rotate_obj_pp(`$\langle obj\rangle$`,`$\langle lpn\rangle_1$`,`$\langle lpn\rangle_2$`,`a`)`: $\langle obj\rangle$ is rotated around an axis going through local points $\langle lpn\rangle_1$ and $\langle lpn\rangle_2$; the rotation is by a degrees.
- `translate_obj(`$\langle obj\rangle$`,`$\langle avn\rangle$`)`: object $\langle obj\rangle$ is translated by vector $\langle avn\rangle$.
- `scale_obj(`$\langle obj\rangle$`,`v`)`: object $\langle obj\rangle$ is scaled by v.

Building new points in space Three macros are especially useful for the definition of regular polyhedra:

- `rotate_in_plane(`k`,`o`,`i`,`j`)`: get point k from point j by rotation around point o by an angle α equal to the angle from i to j; i, j and k are of type $\langle lpn\rangle$, whereas o is of type $\langle apn\rangle$.



- `new_face_point(`c`,`o`,`i`,`j`,`α`)`: the middle m of points i and j is such that $(\widehat{\overrightarrow{om}, \overrightarrow{mc}}) = \alpha$ and $\overrightarrow{mc}$ is $\overrightarrow{om}$ rotated around $\overrightarrow{ji}$. c, o, i and j are of type $\langle lpn\rangle$.



- `new_abs_face_point(`c`,`o`,`i`,`j`,`α`)`: similar to the previous definition, but c and o are of type $\langle apn\rangle$.

Drawing points, axes, objects

- `draw_point(`$\langle obj\rangle$`,`$\langle lpn\rangle$`)`: draw point $\langle lpn\rangle$ in object $\langle obj\rangle$.
- `draw_axes(`$\langle col\rangle_1$`,`$\langle col\rangle_2$`,`$\langle col\rangle_3$`)`: draw vectors $\vec{\imath}$, $\vec{\jmath}$ and $\vec{k}$ in colors $\langle col\rangle_1$, $\langle col\rangle_2$ and $\langle col\rangle_3$.
- `draw_obj(`$\langle obj\rangle$`)`: draw object $\langle obj\rangle$.

Setting faces

- `set_face(`$\langle afn\rangle$`,`$\langle vsl\rangle$`,`$\langle hc\rangle$`)`: set absolute face $\langle afn\rangle$ as delimited by the vertex list $\langle vsl\rangle$ (local point numbers) and colored by color $\langle hc\rangle$.
- `set_obj_face(`$\langle lfn\rangle$`,`$\langle vsl\rangle$`,`$\langle hc\rangle$`)`: set local face $\langle lfn\rangle$ as delimited by the vertex list $\langle vsl\rangle$ (local point numbers) and colored by color $\langle hc\rangle$.

View points, distance

- `compute_reference(`ψ`,`θ`,`ϕ`)`: defines the orientation of the observer by the three angles ψ, θ and ϕ. See figure 5.
- `point_of_view_obj(`$\langle obj \rangle$`,`$\langle lpn \rangle$`,`ϕ`)`: the orientation of the observer is defined as looking local point $\langle lpn \rangle$ of object $\langle obj \rangle$, with an angle of ϕ;
- `point_of_view_abs(`$\langle apn \rangle$`,`ϕ`)`: the observer's orientation is defined as looking absolute point $\langle apn \rangle$, with an angle of ϕ;
- `obs_distance(`v`)(`$\langle obj \rangle$`,`$\langle lpn \rangle$`)`: let v equal the distance between the observer and local point $\langle lpn \rangle$ in object $\langle obj \rangle$.

Vector and point allocation

- `new_vect`$\rightarrow \langle avn \rangle$: return a new vector;
- `new_point`: synonym of `new_vect`;
- `new_points(`v`)(`n`)`: defines the absolute points $v_1, \ldots, v_n$, using `new_point`;
- `free_vect(`$\langle avn \rangle$`)`: free vector $\langle avn \rangle$;
- `free_point(`$\langle apn \rangle$`)`: free point $\langle apn \rangle$;
- `free_points(`v`)(`n`)`: frees the absolute points $v_1, \ldots, v_n$, using `free_point`.

Debugging

- `show_vect(`$\langle str \rangle$`,`$\langle avn \rangle$`)`: shows vector $\langle avn \rangle$, with string $\langle str \rangle$.
- `show_point`: synonym of `show_vect`
- `show_pair(`$\langle str \rangle$`,`$\langle pair \rangle$`)`: this shows a numeric pair, with string $\langle str \rangle$.

Normalization

- `normalize_obj(`$\langle obj \rangle$`,`$\langle vl \rangle$`)`: normalize object $\langle obj \rangle$ with respect to the list of vertices $\langle vl \rangle$.

Parameters

- `Obs_dist` $\rightarrow \langle num \rangle$: distance between the observer and the projection plane.
- `h_field` $\rightarrow \langle num \rangle$: horizontal field of view (default: 100 degrees)
- `v_field` $\rightarrow \langle num \rangle$: vertical field of view (default: 70 degrees)
- `Obs_phi` $\rightarrow \langle num \rangle$: angle ϕ for the orientation of the observer;
- `Obs_theta` $\rightarrow \langle num \rangle$: angle θ for the orientation of the observer;
- `Obs_psi` $\rightarrow \langle num \rangle$: angle ψ for the orientation of the observer;
- `drawing_scale` $\rightarrow \langle num \rangle$: scale factor applied for drawing;
- `filled_faces` $\rightarrow \langle bool \rangle$: if `true`, the faces are drawn filled; if `false`, only the edges are drawn, and hidden edges are drawn dashed;
- `draw_contours` $\rightarrow \langle bool \rangle$: if `true`, the contours of the faces are drawn, and the lines have the thickness `contour_width`; if `false`, the contours are not drawn;
- `contour_width` $\rightarrow \langle num \rangle$: dimension used for drawing contours of faces (default: 1pt).

Constants These values represent constant objects such as reference vectors, and should not be changed.

- `vect_null` $\rightarrow \langle avn \rangle$: internal index for $\vec{0}$.
- `vect_I` $\rightarrow \langle avn \rangle$: internal index for $\vec{\imath}$.
- `vect_J` $\rightarrow \langle avn \rangle$: internal index for $\vec{\jmath}$.
- `vect_K` $\rightarrow \langle avn \rangle$: internal index for $\vec{k}$.
- `point_null` $\rightarrow \langle apn \rangle$: internal index for $\vec{0}$.
- `Obs` $\rightarrow \langle apn \rangle$: observer's internal point number.

Defining new object points and faces

- `new_obj_points(`$\langle obj \rangle$`,`$\langle num \rangle$`)`: defines points 1 to $\langle num \rangle$ in object $\langle obj \rangle$; must be used before setting the points;
- `new_obj_faces(`$\langle obj \rangle$`,`$\langle num \rangle$`)`: defines $\langle num \rangle$ faces in object $\langle obj \rangle$; must be used before setting the faces;

Offsets

- `pnt(`$\langle lpn \rangle$`)` $\rightarrow \langle apn \rangle$: returns the absolute point number for a given local point index.
- `face(`$\langle lfn \rangle$`)` $\rightarrow \langle afn \rangle$: returns the absolute face number for a given local face index.

Standard classes Five standard classes are defined in `poly.mp`: they define the five regular convex polyhedra. For each class $\langle class \rangle$, there are two macros:

- `set_`$\langle class \rangle$`_points` (e.g. `set_cube_points`)
- `def_`$\langle class \rangle$ (e.g. `def_cube`)

Each of these macros is defined with a parameter which is the instance name.

Standard animations The `3d` package provides a few standard animations using the convex polyhedra. In each of these animations, the observer follows a circular path pictured in figure 7. Each standard animation is divided into two macros. The first, such as `animate_object`, defines the class(es) that are used and sets the objects. The second, such as `one_image`, sets the observer, draws the object(s) and moves the object(s) and the observer. The file `animpoly.mp` gives examples of the use of the standard animations.

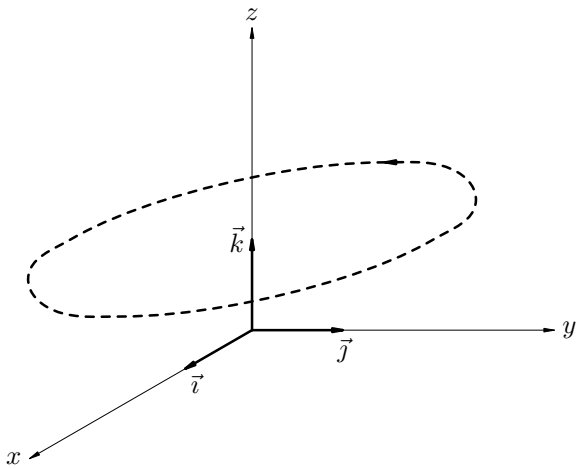


Figure 7: Motion of the observer

Appendix B
Coding an object

In order to extend the `3d` package, it is necessary to understand how the objects are coded. We give here an overview of this coding, but the reader is advised to peek in the code to get a better understanding on how all the functions interact.

First, an object has a name, for instance `"box"`. The macro `box_class` (which can be called with `obj_class_("box")`) is the string corresponding to the class of `"box"`, for instance `"cube"`. The variable `cube_point_offsetbox`, of type `numeric`, and obtained with `obj_point_offset_("box")`, is equal to the absolute index of the last point of the previous object. A cube is defined with $8 + 1$ points. Assuming it was defined after an icosahedron ($12+1$ points) named `"ico"`, `cube_point_offsetbox` will be a `numeric` equal to 13. `cube_pointsbox` (obtained with `obj_points_("box")`) is a macro equal to 9. The variable `cube_face_offsetbox`, similar to `cube_point_offsetbox`, obtained with a call to `obj_face_offset_("box")`, equals 20. The macro `cube_facesbox` (obtained by `obj_faces_("box")`) is equal to 6.

The `obj_name` macro is extended each time a new object is defined. To an absolute face number, it associates an object name. Hence, it is possible to go through all faces. `last_point_offset_` and `last_face_offset_` are the absolute numbers of the last points and faces defined up to now.

```
def obj_name(expr i)=
  if i<1: elseif i<=20:"ico"
          elseif i<=26:"box"
  fi;
enddef;
```

`pnt(i)` gives the absolute vector corresponding to local point i. `ipnt_(i)` is the absolute point number, that is i plus the number of points defined beforehand in other objects. `points_[j]` is the absolute vector corresponding to absolute object point j. Similarly, `face(i)` is the absolute face corresponding to local face i.

The list of vertices of absolute face number i is `face_points_[i]`. The color of absolute face number i is `face_color_[i]`.

When the macros `pnt` or `face` are to be used, the calls `define_current_point_offset_("box")` and `define_current_face_offset_("box")` must be issued.

◇ Denis Roegel
CRIN (Centre de Recherche en Informatique de Nancy)
Bâtiment LORIA
BP 239
54506 Vandœuvre-lès-Nancy
FRANCE
roegel@loria.fr
URL: http://www.loria.fr/~roegel

Font Forum

'Hinting' of scalable outline fonts

Berthold K.P. Horn

'Hinting' refers to methods that guide grid fitting of continuous glyph outlines onto a discrete grid, such as those found on a display screen or laser printer.

The need for hinting

Scalable outline fonts — such as fonts in Adobe Type 1 and TrueType format — have continuous shapes described by mathematical curves. These are used to create a discrete raster of dots on a display or hardcopy output device at a specified size. If such a bitmap is made in a simplistic way — such as simply blackening each cell whose center lies within the contour — then a number of visually distracting artifacts arise — such as misalignments of features and breaks in shapes, also called 'drop-outs.'

Typically hinting is used to do things like:

- make sure stems intended to be equally thick appear equally thick;
- suppress overshoots — rounded letters (O) are taller than flat ones (X);
- line up features on different glyphs that should be at the same height;
- avoid 'drop-outs';
- keep counters between stems open;
- force consistent spacing between sets of parallel strokes;
- compensate for 'misfeatures' of the rasterization algorithm such as drop-outs.

A simple example

A simple demonstration of the need for hinting is the following: suppose you have the letter 'H,' which — when scaled to the discrete grid of the output device — happens to have vertical stems from $x = 1.2$ to $x = 2.8$ and from $x = 4.8$ to $x = 6.4$. So the original width of both stems is 1.6 pixels.

Now suppose that the rasterizer 'inks' every cell whose *center* is inside the outline. Also suppose that the discrete grid of pixels has integer coordinates at the corners and that we round continuous values to the pixel centers. Then the discrete version of the left stem will run from $x = 1$ to $x = 3$, while the discretized version of the right stem goes from $x = 5$ to $x = 6$. In the discrete version then, the left stem is twice as fat as the right stem — a visually very noticeable difference.

Hinting is very important at low resolution (on screen, typically 72 dpi or 96 dpi), where it is hard to get an adequate representation of a continuous shape on a discrete grid. It is still significant at medium (laser printer, typically 300 dpi or 600 dpi) resolution, but is not important at very high resolution (image setter). 'Font smoothing' or anti aliasing also reduces the importance of hinting, although hinting can reduce the number of 'fuzzy edges' that can be distracting in anti-aliased fonts.

Different hinting 'languages'

Intelligent grid-fitting of continuous shapes onto a discrete grid requires a mapping that involves more than simply rounding of x and y coordinates. Much can be achieved by instead setting up a piece-wise linear mapping for each coordinate axis, with corner points at the edges of stems and alignment zones. An alternative to such a 'global' approach is one that depends instead on individually adjusting coordinates of knots. The latter approach is more powerful, but more awkward to use.

The Adobe Type 1 hinting language is declarative, hence easy to use. Since the 'brains' behind it is in the rasterizer, rasterizing quality of a font may be improved if an improved rasterizer is release. However, since the Type 1 rasterizer is not described anywhere, the exact effect of various hinting methods can only be ascertained by experiment.

The TrueType hinting language is imperative and powerful, but also painful to use by comparison. The TrueType rasterizer *is* described, so it is possible — in principle — to predict what the effects of a change might be. One reason the TrueType hinting language *needs* to be more powerful is that the underlying rasterizer has misfeatures (like drop-outs) that need to be 'patched up'. All the 'brains' is in the TrueType font — not in the rasterizer — so it benefits less from improvements to the rasterizer.

METAFONT also provides methods for grid-fitting outlines, as described in 'Discreteness and Discretion,' chapter 24 of *The METAFONTbook*. There is no separate 'hinting language' in this case. The work is done using METAFONT's built-in mechanisms for enforcing constraints established by equalities or inequalities.

Hint switching and delta hints

Quality hinting code is tightly interlaced with the program that draws the glyph, since different 'hints' may apply to different parts of the glyphs. This is called 'hint switching'.

For a simple example, let us consider the 'R' in CMR10. It has a slab lying on the baseline at the

bottom of the left leg and a curled shape that drops *below* the baseline on the right leg. If horizontal stem hints are applied to correctly grid fit the slab on the left leg, then the right leg will be drawn below the baseline, even at very small sizes where this looks 'wrong' and where pulling the right leg up to the baseline produces a visually more pleasing result. Conversely, if a horizontal stem hint were to be tuned for the curl in the right leg, then the slab at the bottom of the left leg would tend to be rasterized floating above the baseline. To get the best rasterization at all sizes, the glyph has to be split into two parts and the right leg rasterizered with different hints than those used for the left.

While simple shapes (like a sans serif 'H') require no hint switching at all, some complex shapes such as the 'Weierstrass p' (℘) may require many switches.

TrueType provides a mechanism called 'delta hinting' which makes it possible to make corrections specific to particular resolutions (measured in ppem — pixels per 'em'). This is useful in correcting drop-outs which occur over certain size ranges. A glyph has to be checked at all sizes that a user is likely to use ('waterfalls'), and drop-outs patched up using delta hints that apply at those sizes. It is possible to draw a complete bitmap using TrueType hinting code.

Type 1 fonts use a different (undocumented) rasterization algorithm that first creates a continuous outline that has no drop-outs, but that is approximately ½ pixel too wide all the way around. It then erodes this outline using Euler-number-preserving binary image operations — which cannot introduce drop-outs. Hence there is no need to 'patch up' the result.

Hint types and hinting tools

Some hints are font wide, such as information on 'alignment zones,' dominant stem widths, and overshoot suppression. But the bulk of hinting code is at the character level. Often the character level code is designed to interact with the font level code (e.g. so-called 'ghost' stems interacting with font level alignment zones at x-height, cap-height, figure-height, ascender, descender heights and so on).

Some simple types of hints follow systematic rules and can be generated automatically by some applications such as Fontographer, FontLab, TypeDesigner, and Ikarus. Others require judgement; quality hinting is more of an art than a science. For example, you get to choose (roughly speaking) between preserving more of the unique character of a particular face at small sizes versus making it more readable.

There are few good tools for doing quality manual hinting. For Type 1 fonts, most font generation software has some support for manipulating hints graphically. Type Designer even provides for hint switching. But without being able to see clearly what hints go where, it isn't really good enough to be useful. For TrueType, the only hinting tools are from Type Solutions in New Hampshire.

The quality of fonts and of rasterizers

Note that the 'hinting' code can be a substantial part of a font, particularly in the case of quality TrueType fonts. If you pass one of the Microsoft core fonts (Arial, Times New Roman, and Courier New) through one of the font generation tools you will see a dramatic drop in size as a result of the fact that the original hints are lost. In some cases quality TrueType fonts shrink by almost 50% (and lose their quality, of course) by this kind of liposuction. Type 1 font hinting is simpler and so rarely adds more than 10% to 20% to file size.

Hinting code is interpreted by the rasterizer and so its effectiveness depends on how well the rasterizer makes use of it. Adobe Type Manager (ATM) does the best job for fonts in Type 1 format, with PostScript interpreters in printers typically not being quite as sophisticated. Various Display PostScript systems mostly do not do as well as ATM either. This may change as Adobe upgrades PostScript interpreters under its control. Naturally 'clones' don't do as well.

Moral

Use only quality fonts. Fonts on those discs that offer 'a zillion fonts for $29.95' are mostly poorly made derivatives of the real thing. Typically the original hinting is stripped out (perhaps because it is thought to be better protected by copyright than the rest of the font program).

References

Adobe Systems Inc. *Adobe Type 1 Font Format.* Addison-Wesley, 1.1 edition, 1993.

Adobe Systems Inc. "Font Hint Information". `http://www.adobe.com/supportservice/devrelations/typeforum/hinting.html`, 1997. Includes references to electronic copies of the Type 1 format specification and related information.

Apple Computer, Inc. *Inside Macintosh/ Text.* Apple technical library. Addison-Wesley, Reading, MA, USA, 1993.

Apple Computer, Inc. "How the Font Manager Renders Outline Fonts". http://devworld.apple.com/dev/techsupport/insidemac/Text/Text-199.html, 1997. From the book *Inside Macintosh/ Text*, chapter 4.

Hersch, Roger D. and C. Betrisey. "Advanced Grid Constraints: Performance and Limitations". In *Raster Imaging and Digital Typography*, edited by R. A. Morris and J. André, pages 190–204. Cambridge University Press, 1991a.

Hersch, Roger D. and C. Betrisey. "Model-based Matching and Hinting of Fonts". *Computer Graphics* **25**(4), 71–80, 1991b. Proceedings of SIGGRAPH'91.

Karow, Peter. "Automatic Hinting for Intelligent Font Scaling". In *Raster Imaging and Digital Typography*, edited by J. André and R. D. Hersch, pages 233–241. Cambridge University Press, 1989.

Karow, Peter. *Digitale Schriften — Darstellung und Formate*. Springer-Verlag, 1992.

Knuth, Donald E. *The METAFONTbook*, volume C of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA, 1986.

Lee Hetherington, I. and P. Tutelaers. "Type 1 utility programs". Available from CTAN fonts/utilities/t1utils, 1992. A set of programs for examining Type 1 fonts.

Penney, Laurence. "TrueType Typography". http://www.truetype.demon.co.uk/tthints.htm, 1997.

Studio Verso. "Font Discussion: Hinting". http://fonts.verso.com/hinting/, 1996. Comments from a W3C discussion of Web fonts in November 1996.

The Microsoft Corporation. "Features of TrueType: TrueType Hinting". http://www.microsoft.com/truetype/hinting/hinting.htm, 1996.

Type Solutions, Inc. "Type Solutions TrueType hinting software: Visual TypeMan and Visual StingRay". http://www.typesolutions.com/TTTools.html, 1997.

◇ Berthold K.P. Horn
Y&Y, Inc.
45 Walden St.
Concord, MA 01742 USA
bkph@ai.mit.edu

Another Approach to Barcodes

Peter Willadt

Abstract

This article copes with barcodes, in particular with interleaved two-of-five and with code 39. It shows various means to generate them with TEX and related software, even the use of VF-files for barcode generation.

1 Background

1.1 Some Common Barcode Types

In stores, upc and ean codes are widely used for automatic identification, pricing, etc. ean barcodes have already been covered in *TUGboat* [1].

In an industrial environment, other barcodes are in use. This article deals especially with two of them, interleaved two-of-five (ITF for short) and code 39.

Original two-of-five code consists of five bars: two of them thick and three thin. With the ten different possible combinations, just the ten digits could be coded. Original two-of-five ignores the width of the gaps between the bars and so it wastes lots of space. One remedy against that was the creation of barcodes where the gaps became meaningful. With ITF, however, there was the somewhat unlucky decision to use the gaps not in a way where one digit would be coded with three bars and two gaps, but to code two digits together — the first digit within the bars and the second one within the gaps (that's where the *interleaved* comes from).

As a consequence, ITF is quite easy to produce by a program, but it is very hard to generate with a text processor or within a report generated from database output. Also you can only code an even number of digits. With quantities, this is no problem, but with fixed-length article codes, things may be different.

The us postnet code is a special kind of two-of-five, where the bars have different height instead of different width. It has already been covered in-depth in *TUGboat* [3, 2].

Code 39 consists of 9 elements — five bars and four gaps. Three of these nine elements are thick, the others are thin (hence *three-of-nine*). As this gives a lot of possible combinations, code 39 can be used not only for digits, but also for uppercase letters from A to Z and seven special signs (+−/ dot, space, dollar, and percent). The bad news about code 39 is that it also takes lots of space. Coding an eight-digit number in medium resolution may easily take two inches of space.

Figure 1: Codabar looks like this. If you have a barcode reader, you should read c1009*

Codabar consists of seven elements — four bars and the three gaps between them. The meaning is contained in two or three thick elements, hence codabar could also be called two-or-three-of-seven. Codabar can be used for the ten digits and $+ - / :$, dot, and dollar. Strangely enough, codabar uses four sets of start/stop-signs. You have to use either the set a/t, b/n, c/* or d/e. The start/stop-signs will be decoded together with the digits between them.

1.2 Usage with TEX

TEX sometimes is used as a back end for database output. I use TEX also for the creation of labels for pharmaceutics for in-house use.

With the ligature mechanism of TEX, ITF output can be easily and transparently handled. And, through TEXs ability to draw bars, also the creation of barcodes without using any other software is possible.

1.3 Readability

Barcodes look rather ugly, so one tends to make them as small as possible. But for readability, there are limits. For a normal scanning device, the width of a narrow bar in code 39 or in ITF must not be less than $7.5 \cdot 10^{-3}$ inches (≈ 0.2 mm). In ITF and code 39, thick gaps or bars have to be at least twice as thick as thin gaps or bars. In higher resolutions, making them 2.25 to 3 times as wide may yield better results.

The bleeding of the bars has not only to be considered for the output device of TEX, but if there is a printing step to follow, the bleeding of the secondary printer has to be compensated, too — at least at higher bar code resolutions. Code 39 and ITF should appear to be about 50% black, not much darker. The contrast between barcode and background should be high. As most reading devices use red light, the bars should not be printed red.

To the left and right of the barcode, there has to be some space in background color. The higher the bars are, the more the reader can be rotated with respect to the bars. If the bars are as high as the complete code is wide, the reader can be twisted at an angle of ± 45 degrees.

It is always best to have access to a reading device to verify good readability.

2 Making Barcodes Work

There are different approaches that fortunately all work. The first approach is to use TEX to draw the barcodes entirely on its own. The great advantage is complete portablity and also an independent control over both height and width of the barcodes. The disadvantage is that macros have to be used for every bar that is to be shown. This may lead to some problems and also to slow run-time behaviour. When coding ITF, the macros get rather complicated.

Another approach would be to use METAFONT to draw the barcode characters. This approach is straight forward. Also with METAFONT there is very fine control over bleeding edges, etc. This control is especially useful with high-density barcodes or with low-quality printers. The disadvantage of using METAFONT is the requirement to create the barcodes at all needed resolutions.

The third approach uses a virtual font (VF-) file to do all the drawing. As a virtual font can be used not only to map from one font or charset to another, but also to draw bars, we get a flexible solution. To TEX the barcodes are just characters from another font; the DVI driver just has to draw rules. The main disadvantage is that we lose the fine control over printer-dependent bleeding, etc., so that I recommend using this solution only for low- to medium-resolution barcodes. Another disadvantage lies in the absolute lack of meta-ness: in VPL-files, everything has to be coded by hand and spelled out explicitly. Any change required results in a lengthy editor session.

2.1 Barcodes by TEX

It is not difficult to draw bars. But drawing barcodes requires drawing a lot of bars. Having defined macros for the obvious start/stop-code and the individual characters, one can draw barcodes by invoking the control sequences directly or — more elegantly — by making characters active and letting them draw themselves. The third solution — reading the characters to draw as parameters to a macro — works well for a numerical-only barcode, but 43-way branching as would be necessary for code 39 is surely not attractive. Here is a short excerpt from the coding of code 39.

```
\newdimen\dick     \newdimen\dickbar
\newdimen\duenn    \newdimen\duennbar
\newdimen\antibleed
\dick=1.2mm        \duenn=0.6mm
\antibleed=0mm
\dickbar=\dick     \duennbar=\duenn
```

Figure 2: Code 39 looks like this. These lines mean *HELLO*

```
\advance\dickbar by -\antibleed
\advance\duennbar by -\antibleed
   % b means bar...
\def\b{\vrule width\duennbar}
\def\B{\vrule width\dickbar}
   % ...and s means space
\def\s{\hskip\duenn\hskip\antibleed}
\def\S{\hskip\dick\hskip\antibleed}
   % and then the coding
\def\tninestart{\b\S\b\s\B\s\B\s\b\s}
\def\tnineminus{\b\S\b\s\b\s\B\s\B\s}
\def\tninelettera{\B\s\b\s\b\S\b\s\B\s}
\def\tnineleletterb{\b\s\B\s\b\S\b\s\B\s}
   % many lines of code omitted
\def\makethemactive{%
    \catcode`\A=\active
    \catcode`\B=\active
   % many lines of code omitted
}
{\makethemactive
 \gdef\begincodethirtynine{%
    \bgroup\makethemactive
    \strut\tninestart
    \let A=\tninelettera
    \let B=\tnineletterb
   % many lines of code omitted
 } % end begincodethirtynine
}  % end scope makethemactive
\def\endcodethirtynine{%
    \tninestart\egroup
}
```

The text to be printed as a barcode can then be included between `\begincodethirtynine` and `\endcodethirtynine`.

The code width and height can easily be adjusted from TEX, and the extra blackness added by the printing engine can also be removed by adjusting `\antibleed`. The coding is somewhat fuzzy, because many characters, even the space, have to get a special meaning already where `\begincodethirtynine` is being defined.

If only digits have to be printed, a multiway branch may be suitable to draw the bars and the coding can be handled by tail recursion. The following example shows the coding of a *Pharmazentralnummer*. These are article numbers for german pharmaceuticals. They consist of exactly seven digits; in code 39, they are preceded by a minus sign. The last digit is a weighted mod 11 checksum. In the following example, besides drawing bars, TEX also counts the digits and calculates the checksum.

```
% needs the code printed above
% three counts for checksumming etc.
\newcount\ziffern
\newcount\checksum
\newcount\multreg
% numbers in code 39
 \def\tnzero{\b\s\b\S\B\s\B\s\b\s}
  \def\tnone{\B\s\b\S\b\s\b\s\B\s}
  \def\tntwo{\b\s\B\S\b\s\b\s\B\s}
\def\tnthree{\B\s\B\S\b\s\b\s\b\s}
 \def\tnfour{\b\s\b\S\b\s\b\s\B\s}
 \def\tnfive{\B\s\b\S\B\s\b\s\b\s}
  \def\tnsix{\b\s\B\S\B\s\b\s\b\s}
\def\tnseven{\b\s\b\S\b\s\B\s\B\s}
\def\tneight{\B\s\b\S\b\s\B\s\b\s}
 \def\tnnine{\b\s\B\S\b\s\B\s\b\s}
\def\tndigit#1{%
  \ifcase#1\tnzero\or\tnone
    \or\tntwo\or\tnthree
    \or\tnfour\or\tnfive
    \or\tnsix\or\tnseven
    \or\tneight\or\tnnine
    \fi%
}
\def\endtncode{%
  \ifnum\ziffern=9
    \else\message{wrong digits count}%
  \fi
  \ifnum0=\checksum
    \else\message{wrong checksum}%
  \fi
  \tninestart\egroup
}
\def\nexttn#1{%
  \advance\ziffern by1
  \if@#1\let\next\endtncode
   \else
   \tndigit#1
% begin checksum stuff
   \ifnum\ziffern=8
    \multreg=\checksum
    \divide\multreg by 11
    \multiply\multreg by 11
    \advance\checksum by-\multreg
    \multreg=#1
    \advance\checksum by-\multreg
    \ifnum\checksum=10
        \checksum=0
    \fi
   \else
    \multreg=#1
```

```
    \multiply\multreg by\ziffern
    \advance\checksum by\multreg
   \fi
  \fi
% end checksum stuff
  \next
}
\def\pzncode{%
  \bgroup
  \let\next\nexttn
  \ziffern=1\checksum=0\multreg=0
  \strut
  \tninestart\tnineminus%
  \next
}
% Example (right)
\pzncode1234562@
% Example (wrong)
\pzncode1235462@
```

2.2 Barcodes by METAFONT

The advantages of METAFONT are quite clear: It is easy to draw bars, to build characters out of them, to cope with the printer's bleeding and similiar effects (the german word is *Druckzuwachs*), and, because of the meta-ness, very compact and flexible code can be written. When the bars have to be combined with other elements (ean code, e.g., requires the number printed in ocr under the bars) METAFONT is the first choice.

Using pk fonts, the fonts produced are rather compact — they may be even shorter than the corresponding VF-files.

As this article already tends to be lengthy, I will not treat METAFONT any further.

2.3 Barcoding with virtual fonts only

Virtual fonts can contain any instruction that is also found in a DVI file. So, besides typesetting letters and moving, virtual fonts can also contain instructions to draw bars. As an example, I have completely mapped ITF barcodes into a virtual font. The handling of two digits sharing the same bars and gaps is left entirely to the ligature mechanism, so the font contains a little more than a hundred characters.

The header of the VPL file looks like the header of any VPL file for a monospaced, upright font.

```
(FAMILY BARCODE)
(DESIGNSIZE D 12)
(DESIGNUNITS D 14)
(COMMENT written by Peter Willadt)
(COMMENT August 16, 1997)
(FONTDIMEN
   (SLANT   R  0)
   (SPACE   D 14)
   (SHRINK  D  0)
   (STRETCH D  0)
   (XHEIGHT R 10)
   (QUAD    D 14)
   )
```

Everything quoted can be found in the documentation for VPtoVF, so I will not discuss it in detail. Just let me mention that the design size of the font is twelve points, and that these twelve points are divided into fourteen units (five bars and five gaps, and two thick bars and gaps each). Thin units will be about 0.3 mm; that makes a medium resolution.

After the heading, there follows an extensive `ligtable`, where every combination of each digit with any other digit is mapped to a character. Here is a short excerpt, showing the zero preceding any digit:

```
(LIGTABLE
   (LABEL C 0)
   (LIG   C 0 D 1) (LIG   C 1 D  2)
   (LIG   C 2 D 3) (LIG   C 3 D  4)
   (LIG   C 4 D 5) (LIG   C 5 D  6)
   (LIG   C 6 D 7) (LIG   C 7 D  8)
   (LIG   C 8 D 9) (LIG   C 9 D 10)
   (STOP)
```

The rest is too dull to show here. The characters from 48 to 58 have not been used for ligatures; this not only gives clear code, but also avoids reentering the ligature mechanism.

Start- and stop-codes are different for ITF. I have mapped them to the + and − keys. Codings for 0–9 themselves are contained in the font, because there is no possibility to have ligatures when the constituent characters do not exist. They only get drawn when there is an odd number of digits to code; this is a case that should never happen. So there are two possibilities: make the mistake immediately visible (e.g. by just skipping instead of drawing) or make the best of it. I chose the latter way and mapped these codes to the same chars that would result from the digit drawn twice — hoping that an excess digit at the end would do no harm.

Let us just have a short look at the character that will be drawn when a 0 is followed by a 5:

```
(CHARACTER D 6
   (COMMENT 0 and 5)
   (CHARWD D 14)     (CHARHT R 14)
   (CHARDP R  0)     (CHARIC R  0.0)
   (MAP
      (SETRULE R 14 R 1)(MOVERIGHT R 2)
```

Hello, the plate with the mix computer's serial number looks like this:

Figure 3: ITF code. Output of the sample code.

```
    (SETRULE R 14 R 1)(MOVERIGHT R 1)
    (SETRULE R 14 R 2)(MOVERIGHT R 2)
    (SETRULE R 14 R 2)(MOVERIGHT R 1)
    (SETRULE R 14 R 1)(MOVERIGHT R 1)
    )
)
```

All the action consists of drawing several bars, each 14 units (12 pt) high, and 1 or 2 units wide, and skipping 1 or 2 units after every bar.

VPL files are human-readable, but I guess they were not intended to be human-writable. You have to painfully avoid using the tab-key for indenting; also, it is recommended to indent in a very intentional way, lined up as shown in the example, and with every level of indentation shifted by the same number of spaces further to the right.

Having the VPL file, you simply have to make TFM and VF files out of it. You do this by invoking VPtoVF and copying the resulting files to locations where TEX and the printer driver can find them.

Within TEX, ITF can then simply be used like any other font; it looks like this:

```
\font\itf=wlitf scaled 2000
\def\itfcode#1{{\itf+#1-}}
Hello, the plate with the
mix computer's serial number
looks like this:
\centerline{\itfcode{1009}}
```

Higher TEXnique would require checking whether the argument to `\itfcode` is completely numeric and if it consists of an even number of digits — maybe even calculating a checksum. But for processing database output, the checking should happen at an earlier stage.

Because of the possibility to include references to other fonts within a VPL file, it is also possible to build a virtual font containing bars and digits — something that you need for upc codes etc. — provided you have already got a font of ocr digits.

3 Conclusion

You can create barcodes any way you like; TEX offers several opportunities. The conventional method of using METAFONT is surely the best way, but for one-time-use TEX itself may suffice. Using a VPL-file alone results in scalable fonts, but without hinting.

The files related to this article (code 39 both in METAFONT and TEX macros, an ITF font and a codabar font, both in VPL format, and an ean font in METAFONT format) can be found on CTANin `../tex-archive/fonts/barcodes/willadt/`. You should be aware of the fact that for the METAFONT code, you need to do some preprocessing. All fonts and macros may be freely used without restrictions.

References

[1] Peter Olšak. The EAN barcodes by TEX. *TUGboat*, 15(4):459–464, 1994.

[2] John Sauter. Postnet codes using METAFONT. *TUGboat*, 13(4):472–476, 1992.

[3] Dimitri Vulis. TEX and envelopes. *TUGboat*, 12(2):279–284, 1991.

◇ Peter Willadt
Heinrich-Wieland-Allee 5
75177 Pforzheim
Germany
`Willadt@t-online.de`

Abstracts

Die TEXnische Komödie
Contents of Some Past Issues

6. Jahrgang, Heft 1/1994 (Juli 1994)

Luzia Dietsche, Editorial; p. 3

Two short comments by the editor: The last editorial discussing spaces preceding punctuation marks have caused considerable reaction. The common view: Rules have not changed, additional inserted spaces are (at most) trendy. And in the process of phototypesetting the last issue a figure has been reproduced too small (change of resolution). Therefore, the whole article is reprinted (see pp. 29–33).

◦ *Hinter der Bühne : Vereinsinternes*
[Backstage : Club matters]; pp. 4–22:

Joachim Lammarsch, Grußwort
[Welcome message]; pp. 4–6

A short comment on club matters by the president of +DANTE+. In particular, he explains that the current issue has been delayed by a protest

issued against the members' assembly (the protocol of which is included on pp. 6–22). MicroPress has mailed p.r. material to all members of +DANTE+, obviously making (illegal) use of the list of members of +DANTE+.

[Luzia Dietsche],
Protokoll der 10. Mitgliederversammlung von +DANTE+, Deutschsprachige Anwendervereinigung TEX e.V. [Protocol of the 10th assembly of members of +DANTE+]; pp. 6–22

This is the official report of the members' meeting held in Münster (Feb. 17, 1994). As usual, it contains short accounts on the various hardware platforms, `german.sty` and other special topics (as presented by the appointed coordinators), followed by a report on the situation of +DANTE+ (mainly organizational matters), and its activities (distribution of books and software, servers, including plans for the future, i.e., a +FAQ+ and a CD), on (partially difficult) contacts with publishing houses (Addison-Wesley, Springer, Vieweg) and the contacts with +TUG+. It is reported that *DANTE Ltd.*, a new European organization, has assured that their field of interest does not coincide with +DANTE+ e.V. Finally, the discussion on a protest claiming manipulation of an election (in Chemnitz) is summarized. (This discussion caused the protest against the assembly mentioned by Joachim Lammarsch, p. 5).

◦ *TEX-Theatertage*
[TEX theatre festival]; pp. 23–27:

Volker Thewalt, +DANTE+'94 in Münster – Ein kurzer Tagungsbericht [+DANTE+'94 in Münster – A short meeting report]; pp. 23–27

A personal account of +DANTE+ '94 (Feb. 16–18, 1994) in Münster.

◦ *Von fremden Bühnen* [On other stages]; p. 28:

Don Hosek, Writers on type and typography needed (in English); p. 28

An advertisement calling for contributions for *Serif*, a new journal on all aspects of typography.

◦ *Bretter, die die Welt bedeuten*
[The stage is the world]; pp. 29–45:

Andreas Scherer, Graphiken mit *GnuPlot* und Metafont–Korrektur [Graphics with *GnuPlot* and Metafont–Correction]; pp. 29–33

This article was published in the previous issue (4/1993, pp. 26–30). It is reprinted because the figure was reproduced incorrectly (much too small): The author briefly describes how GnuPlot can be used to produce Metafont code for line graphics which then can be included in a TEX file as a font.

Michael Baas, TEX-Makros für Fortgelaufene [TEX macros for the runaways]; pp. 34–40

A lively account of the author's efforts to develop (imperfect, but working) macros which simplify the use of active characters for easy input of accented characters (and similar macros). (See also the letter by David Kastrup (issue 6/3, 49–52) for comments and suggestions.)

Arne W. Steuer, Postkarten mit PasTEX auf dem Amiga [Doing postcards with PasTEX (on the Amiga)]; pp. 40–41

The author describes his `postcard.sty` which can be used to print postcards. (For the inclusion of graphics, PasTEX is used.)

Richard Hirsch, Dezimalkomma beim TEXsatz in deutsch [The decimal comma with German TEX typesetting]; pp. 42–45

The German replacement for the decimal point is the decimal comma. However, if one writes `$3,14$` the spacing is wrong while the correct input `$3{,}14$` is inconvenient to use. The author describes in detail how to define two types of commas (with `\mathchardef`) and set up an 'intelligent comma' in mathematics mode (with `\mathcode‘,="8000` which, however, is misprinted as `\mathchardef‘,="8000`) which inserts an ordinary comma if followed by a space, and a decimal comma if not.

◦ *Was Sie schon immer über TEX wissen wollten ...* [What you always wanted to know about TEX ...]; pp. 46–47:

Luzia Dietsche, Seitenzahlen
[Page numbers]; pp. 46–47

Three short hints for LATEX users: centered page numbers, the total number of pages, and page numbers together with the total number of pages, by redefinition of `\thepage`. (Cf. the comments in the next issue 2/94, 50–52.)

◦ *TEX-Beiprogramm* [TEX co-features]; pp. 48–51:

Jörg Lehrke, LinuX – ein freies *nix für PCs [LinuX – a free *nix for PCs]; pp. 48–51

A pointer to and a short description of LinuX 1.0, a full implementation of Unix for PCs, initiated by Linus Torvalds in 1991 — well suited for TEX, but not only for TEX.

◦ *Rezensionen* [Reviews]; pp. 52–53:

Oliver Hansen, Mut zur Typographie [Dare typography]; pp. 52–53

A review of *Mut zur Typographie* by J. Gulbins and C. Kahrmann (Springer 1993), a book explaining the basics of typography and layout. The bottom line: 'Whoever wants to do serious +DTP+, or to do a thesis with a word processor, should take a thorough glimpse (at least) into this book.'

◦ *Leserbrief(e)* [Letter(s)]; pp. 54–55:

Two short letters to the editor. The first (by Martin Schröder) points out two more sources for the permille symbol not mentioned in the article by Andreas Scherer (issue 3/93). The second (from Philipp Wolfrum jr.) asks for help to find some tools for TEX under OS/2.

◦ *Spielplan* [Repertory]; pp. 56–63:

The international and national calendar, and an announcement of a conference (EuroTEX '94).

◦ *Adressen* [Addresses]; pp. 64–67:

Various addresses related to +DANTE+ and +TUG+, the addresses of all persons who have contributed to this issue, and the addresses of the coordinators in charge of the various hardware platforms and other special topics.

6. Jahrgang, Heft 2/1994 (September 1994)

Luzia Dietsche, Editorial; p. 3

The editor hopes that in the future the delay in publication of issues will be reduced.

◦ *Hinter der Bühne : Vereinsinternes* [Backstage : Club matters]; pp. 4–10:

Joachim Lammarsch, Grußwort [Welcome message]; pp. 4–5

A short comment on club matters by the president of +DANTE+. In particular, he explains that the presidium has decided to terminate participation in the +TUG+ Board of Directors.

Jürgen Günther, Aus der Verwaltungsküche [From the administrative kitchen]; pp. 5–7

The article explains how orders are processed and why delivery takes 4–6 weeks on average.

Joachim Lammarsch, Fonds zur Unterstützung von Mitgliedern [Funds for the support of members]; pp. 7–8

How to apply for exemption from membership dues.

Andreas Schrell, Der Stammtisch in Wuppertal [The local meetings in Wuppertal]; pp. 8–10

An account on the the local meetings in Wuppertal (Aug. 92–Dec. 93) by their initiator and organizer. A yearbook of the proceedings is available.

◦ *Von fremden Bühnen* [On other stages]; pp. 11–13:

Hermann Zapf, Has type design any future? (in English); pp. 11–13

This is a widely circulated protest (revised version by Hermann Zapf and Charles Bigelow, +RIDT+'94) against piracy of typefaces.

◦ *Bretter, die die Welt bedeuten* [The stage is the world]; pp. 14–36:

Bernd Raichle, TEX 3 von Version 3.0 bis 3.1415 [TEX 3 from version 3.0 to 3.1415]; pp. 14–26

A description of the changes and the major bug fixes implemented in TEX and `plain` versions 3.0–3.1415, and how these versions can be identified by macros.

Walter Schmidt, Umsteigen auf LATEX2_ε [Changing to LATEX2_ε]; pp. 26–31

A short description of LATEX2_ε (distributed since June 1994), its installation, and its compatibility with LATEX 2.09.

Walter Schmidt, LATEX für Techniker [LATEX for technicians]; pp. 32–36

Standard LATEX lacks some features needed for texts in physics and engineering. The author describes how they (italic greek letters, some symbols, boldface variables) can be implemented with LATEX2_ε.

◦ *TEX-Beiprogramm* [TEX co-features]; pp. 37–46:

Horst Szillat, Internet-Anschluß – selbst gebastelt [Internet connection – do it yourself]; pp. 37–39

A brief overview on the options for private net access in Germany.

Michael Schank, TEX und OS/2 2.x [TEX and OS/2 2.x]; pp. 39–42

A short description of emTEX (`tex386b11`) and GNU emacs with AucTEX for OS/2 (in the implementation by Eberhard Mattes), updating and supplementing an earlier article (issue 5/4).

Markus Porto, TEX-Service des +HRZ+ der Universität Gießen [TEX support by the computer centre of the University of Gießen]; pp. 43–46

At the address `www.uni-giessen.de` the University of Gießen offers an +HTML+-based help and information system on TEX (mainly in German). It includes an interactive *LATEX cookbook* containing code samples and facilities for testing them (or variants of them) online.

◦ *Rezensionen* [Reviews]; pp. 47–49:

[Dr.] Rainer Schöpf, Zwei neue Bücher
[Two new books]; pp. 52–53

Short reviews of *LATEX zum Loslegen* [*Get Going with LATEX*] by Joachim Lammarsch and Luzia Dietsche, and *TEX, LATEX und Graphik* by Friedhelm Sowa (both published by Springer 1994).

◦ *Leserbrief(e)* [Letter(s)]; pp. 50–52:

Frank Mittelbach, Seitenzahlen
[Page numbers]; pp. 50–52

The author points out some side effects of the macros proposed by Luzia Dietsche in the previous issue (1/94, 46–47).

◦ *Spielplan* [Repertory]; pp. 53–55:

The international and national calendar, and the invitation to the autumn meeting of +DANTE+ in Katlenburg-Lindau (Oct. 13–14, 1994).

◦ *Adressen* [Addresses]; pp. 56–59:

Various addresses related to DANTE, the addresses of everyone who has contributed to this issue, and the addresses of the coordinators in charge of the various hardware platforms and other special topics.

6. Jahrgang, Heft 3/1994 (Januar 1995)

Luzia Dietsche, Editorial; p. 3

Two remarks by the editor: Contributions submitted to the *Komödie* tend to form clusters of related material. Also she would like to install a 'bulletin board' for short questions, news, etc. — if she receives such material.

◦ *Hinter der Bühne : Vereinsinternes*
[Backstage : Club matters]; pp. 4–17:

Joachim Lammarsch, Grußwort
[Welcome message]; pp. 4–5

A short comment on club matters by the president of +DANTE+. In particular, he complains about raised bank fees for payments to +DANTE+. *4AllTEX*, the plug&play +CD+ of +NTG+, can be obtained from Dante.

[Luzia Dietsche],
Protokoll der 11. Mitgliederversammlung von +DANTE+, Deutschsprachige Anwendervereinigung TEX e.V. [Protocol of the 11th assembly of members of +DANTE+]; pp. 6–17

This is the official report on the members' meeting held in Katlenburg-Lindau (October 13, 1994). It starts with short accounts on the various hardware platforms, `german.sty` and other special topics (as presented by the appointed coordinators, supplementing a more detailed written report, see pp. 31–45), followed by a report on the situation of +DANTE+ (mainly organizational matters), and its activities. There are some difficulties in +DANTE+'s relation with +TUG+ (payment of membership dues). Joachim Lammarsch has resigned from the Board of Directors. Finally, the status of NTS and ε-TEX is described, and a letter of thanks from Philip Taylor (from the NTS group) is presented.

◦ *Von fremden Bühnen*
[On other stages]; pp. 18–20:

[The LATEX Programming Team], LATEX News, Issue 2, December 1994 (in English); pp. 18–20

This is the *Newsletter* accompanying the second release (Dec. 94) of LATEX2ε. There are two new packages included: `inputenc` (which allows choosing an 8-bit input encoding), and +AMS+-LATEX.

◦ *Bretter, die die Welt bedeuten*
[The stage is the world]; pp. 21–30:

Christian Kayssner, Handbücher
[Manuals]; pp. 21–24

The author describes his program `double` that rearranges the pages of a `dvi`-file for easy 2-up printing of booklets.

Arne W. Steuer, Style-Files – leicht gemacht
[Style files – made easy]; pp. 25–30

Taking labels for organizers as an example, the author shows in detail how (even) a LATEX beginner can develop simple style files.

◦ *TEX-Beiprogramm* [TEX co-features]; pp. 31–45:

Bericht des technischen Beirats
[Report of the technical council]; pp. 31–45

A collection of short reports (by the appointed coordinators) on the various hardware platforms and other special topics, supplementing those of the

protocol (p. 5). One of them (by Joachim Schrod) concerns the working groups on driver standards and directory structure.

◦ *Leserbrief(e)* [Letter(s)]; pp. 46–55:

Andreas Ringhandt, Von A4 zu A5? [From A4 to A5?]; pp. 46–47

This letter asks for advice on how to handle a document (a thesis) that will be printed both in A4 and A5. No answer is given.

Stefan Breuer, ftp-Server, +CD-ROM+ oder Disketten? Die Verteilermedien für TEX [ftp-server, +CD-ROM+ or floppy disks? Media for distributing TEX]; pp. 47–49

The writer argues that a +CD-ROM+ copy of +CTAN+ would be the best way to distribute TEX for users without net access. For information on the contents of +CTAN+ he offers a disk containing the +CTAN+ file `FILES.byname` in text and dBase format.

David Kastrup, TEX-Makros für Fortgelaufene in *Die TEXnische Komödie* 1/1994 [TEX macros for the runaways in *Die TEXnische Komödie* 1/1994]; pp. 49–52

The author comments on the macros described in the article cited in the title (by Michael Baas, issue 6/1, 34–40) from an expert's point of view and suggests some improvements.

Heinz Kusznier, Aus der Verwaltungsküche in *Die TEXnische Komödie* 2/1994 [From the administrative kitchen in *Die TEXnische Komödie* 1/1994]; p. 53

In reaction to the article cited in the title (by Jürgen Günther, issue 6/2,5–7), the writer thanks the +DANTE+ team for their good services.

Martin Schröder, Gedanken zu Gedankenstrichen [Thoughts on dashes]; pp. 53–55

The writer points out that dashes are used differently in German and in English typography. A dash (= *Gedankenstrich*) in German text should be an en-dash between spaces. He criticizes that these differences often are neglected (e.g. in the article by Hermann Zapf as printed in issue 6/2, 11–13) and mainly blames Kopka as the source of this habit.

◦ *Spielplan* [Repertory]; pp. 56–58:

The international and national calendar, and invitation to a conference, +DANTE+'95 in Gießen (Feb. 28–March 3, 1995).

◦ *Adressen* [Addresses]; pp. 60–63:

Various addresses related to DANTE, the addresses of everyone who has contributed to this issue, and the addresses of the coordinators in charge of the various hardware platforms and other special topics.

6. Jahrgang, Heft 4/1994 (Februar 1995)

Luzia Dietsche, Editorial; p. 3

This rather thin issue contains the first article of a new series by Bernd Raichle (*Oral games with TEX*).

◦ *Bretter, die die Welt bedeuten* [The stage is the world]; pp. 4–27:

Gerd Neugebauer, *BibTool* – Manipulation von *BibTEX*-Dateien [*BibTool* – manipulation of *BibTEX* files]; pp. 4–11

'Once upon a time I had the urgent desire to use BibTEX files from various sources in combination.' This desire had consequences: *BibClean*, a program written in C and distributed as source only, is a highly configurable tool which can combine (and unify) BibTEX databases, generate keys, sort them, and extract data using information from `.aux`-files, as well as perform some other manipulations — it is a 'Swiss army knife for handling BibTEX', as the author expresses it. Bernd Raichle has added a pointer to `bibclean` by Nelson Beebe, a prettyprinting and syntax checking program for BibTEX files.

Walter Schmidt, Computer Modern Bright; pp. 11–16

Computer Modern Sans Serif is well suited to emphasize single words or short pieces of text, but less so as the main text font. To fill this gap, the author has developed Computer Modern Bright. This font is obtained from the Computer Modern fonts by a new combination of the parameters and slight corrections to a few letters concerning the dots on the letters i and j and the accents for the German umlauts. The fonts can already be used, but are not yet finished.

Bernd Raichle, Orale Spielereien mit TEX – Teil I [Oral games with TEX – part I]; pp. 16–23

This new series will discuss various aspects of TEX's *mouth*, i.e., its macro processor. The first installment is devoted to the first problem in Appendix D ('dirty tricks') of *The TEXbook*: Define a macro that takes an integer n as input and

expands to a string of n asterisks. First the two solutions described by Knuth and three solutions from *TUGboat* are discussed. Finally a new (December 1993 on `comp.text.tex`) solution based on `\romannumeral` by David Kastrup is presented.

Matthias Eckermann, Paralleles Setzen längerer Texte [Parallel typesetting of lengthy texts]; pp. 23–27

The author describes his (still fragmentary) LaTeX style `parallel.sty` which allows parallel text (e.g., a text and its translation) with automatic page breaks.

◦ *Was Sie schon immer über TeX wissen wollten ...* [What you always wanted to know about TeX ...]; pp. 28–30:

Luzia Dietsche, Absatzformen [Paragraph shapes]; pp. 28–30

LaTeX code centering the last line of a caption or a paragraph, based on setting `\leftskip=0pt plus 1fil`, `\rightskip=0pt plus -1fil`, and `\parfillskip=0pt plus 2fil`.

◦ *TeX-Beiprogramm* [TeX co-features]; pp. 31–36:

Horst Peiffer, WinWord versus LaTeX; pp. 31–34

The author (who classifies himself as a 'TeX-oholic') argues that WinWord convinces beginners by first impressions, and that — though it is not at all easy to use for complex applications — difficulties must pile up considerably before a Word user thinks of switching to TeX.

Michael Schank, TeXIT! for OS/2; pp. 34–35

TeXIT! is the author's OS/2 shell for emTeX (available as beta version).

Luzia Dietsche, Rechtschreibreform [Orthography reform]; pp. 35–36

A parody on efforts to 'simplify' German spelling. (The article is taken from an unknown internet source.)

◦ *Rezensionen* [Reviews]; pp. 37–39:

Harald Schoppmann, TeX für Heimwerker... [Do it yourself TeX...]; pp. 37–39

A review of *Making TeX Work* by Norman Walsh (O'Reilly, 1994). The bottom line: This is not a book for *ordinary* users but for system managers (who have to maintain TeX on several platforms) or a PC user looking for special purpose software or special fonts.

◦ *Spielplan* [Repertory]; pp. 40–43:

The international and national calendar, and the invitation to a conference, the 16th Annual Meeting of TeX Users Group 'Real World TeX' in St. Petersburg Beach, Florida (July 24–28, 1995).

◦ *Adressen* [Addresses]; pp. 44–47:

Various addresses related to DANTE, the addresses of everyone who has contributed to this issue, and the addresses of the coordinators in charge of the various hardware platforms and other special topics.

(compiled by Peter Schmitt)

Les Cahiers GUTenberg
Contents of Issues 26 and 27

Numéro 26 — mai 1997

The pervasiveness of the Internet today is such that it hardly warrants special mention — hard to realise that GUTenberg's own list (`gut@ens.fr`) has been in operation for over ten years now. And we probably haven't even begun to realise its possiblities. The "journées GUTenberg" continue to prove that TEX users are a dynamic component in Internet developments. We can view `.dvi` files via browsers, click on hyperlinks, move from document to document, add colour where we want ... Conversion utilities make it easier to produce Web pages from LATEX documents, and HTML, while itself providing nowhere near the typographic quality of TEX, does seem to be moving more and more into a kind of close symbiotic relationship with LATEX.

Recent efforts to make (LA)TEX more accessible and flexible are bearing fruit: the `tetex` CD for easier installation on UNIX and other platforms has been around for a year now; `Web2C`, together with the TDS standard, is bringing uniformity and simplicity to the general distribution of (LA)TEX ... these are just some of the tremendous international efforts by dedicated volunteers.

The number of applications using TEX and LATEX 2_ε are such that they have passed the point of being simple 'projects' — they're now integral parts of what's happening on the 'Net. And we still have so much more to look forward to: Ω, ε-TEX, LATEX3.

Jacques André, Caractères numériques : introduction [Numeric characters: introduction]; pp. 5–44

This is part of a tutorial on fonts. First, the evolution from hot metal types to digitized characters is shown. Then, the main concepts used in digital typography (bitmaps, outlines, hints, Multiple Masters, etc.) are explained.

[Author's abstract]

[Big bibliography worth consulting — with references in French and English (British and American).]

Thierry Bouche, *Minion* MM : installer une famille de fontes *multi-master* [*Minion* MM: installing a multiple master font family]; pp. 45–70

A rather precise description of the installation procedure of a multiple master text font family like *Minion* MM is given.

[Author's abstract]

Denis Roegel, Les formats de fichiers DVI, GF, TFM et VF : que contiennent-ils et comment les visualiser ? [File formats DVI, GF, TFM and VF: what they contain and how to view them]; pp. 71–95

A normal TEX distribution is made of a large number of programs interacting via files in various formats. We commonly manipulate several of these formats without knowing their contents: DVI, GF, TFM, VF, etc. We examine here in a more detailed manner the contents of these files, using programs that are included in the standard distributions.

[from the author's Abstract]

Fabrice Popinau, Rapidité et souplesse avec le moteur `web2c` 7 [Speed and flexibility with `web2c` 7]; pp. 96–108

[Following a brief overview of `web2c`, the article focuses on some specific elements: `kpathsea`, ancillary programs, and current ports to systems other than UNIX.]

Christophe Prud'homme, Comparaison LaTeX2HTML, HyperLaTeX [Comparing LaTeX2HTML with HyperLaTeX]; pp. 109–120

We present several aspects of two LaTeX-to-HTML conversion programs, including points where they differ or are similar. As well, some possible extensions for each are discussed.

[translation of author's Résumé]

Phil Taylor, Présentation du projet ε-TEX [The ε-TEX project]; pp. 121–132

After a few years of discussions and implementations, the first version of ε-TEX was released in 1996. A lot of new features were added to TEX to increase its functionality. A second version is currently being designed.

[Author's abstract]

Hélène Richy, Feuilles de style pour le Web [Styles files for the Web]; pp. 133–145

Will style files become a required element of the Web? While no-one has the answers just yet, we examine here a "new" approach and its consequences with respect to the Web and *Cascading Style Sheets*.

[translation of author's Résume]

Numéro 27 — juillet 1997

Jacques André, Éditorial : documents électroniques et qualité typographique [Editorial: electronic documents and typographic quality]; pp. 3–4

[GUTenberg, as an association, has always had two main goals: to promote the use of TEX amongst French-speaking users, and to encourage technical exchanges that expand printing and scientific publishing opportunities — the Internet and the Web fall squarely into this latter arena. However, straddling both of these goals is the issue of quality: of fonts, composition, character encoding, typographic norms, and so on.

The two articles in this issue, translations from English, argue for quality, particular when texts are to eventually find their way onto the net. And a final editorial note: that it is interesting to see that some of the problems raised are targetted by ε-TEX.]

Conrad Taylor, Mais qu'est ce qu'ont bien pu nous apporter les systèmes WYSIWYG? [What has WYSIWYG done to us]; pp. 5–33

This paper is a translation of the paper: Conrad Taylor, "What has WYSIWYG done to us", *Seybold Report*, volume 26(2), 30 September 1996, pp. 1–12, and was translated and printed in *Cahier* 27 with permission.

[Translated from footnote 1, p. 5.]

[From the editorial, the following comments serve as a useful summary:
... Taylor questions the predominance of WYSIWYG systems, and replays the arguments of their proponents and critics, in light of ten years' professional experience. He dwells on the difficulty of doing proper page composition with commerical interactive programs, and practically praises programs which unite the structuralism of SGML and the quality of TEX. His great fear: that the companies currently fighting for dominance in Web publishing pretend that there's no further need to address typographic quality.]

Peter Karow, Le programme *hz* : micro-typographie pour photocomposition de

haut niveau [*hz*: micro-typography for advanced typesetting]; pp. 34–70

In the fall of 1992, URW completed computer programs for manufacturers of setting and composition equipment. These programs are outstanding in the field of micro-typography. To express his delight over the results, Hermann Zapf, the inventor, has called them "nonplusultra" in advanced typography. At the beginning, he and URW wanted to rediscover, in the age of computers, that which Johannes Gutenberg had achieved five hundred years ago: namely, a justified setting of text with equal inter-word spacing and optically straight aligned margins. Our *hz*-program has achieved this and even more: its application saves 3–5% of paper, thereby contributing to environmental protection via typographic means.

[from author's abstract]

[From the editorial, the following comments serve as a useful summary:
... Karow addresses mediocre fonts used in document exchange not by shouting "they did it better with lead" but by demonstrating a program which does as well as, if not better than, what was achieved in Gutenberg's day — and on our computer screens, no less.

[The article is based on several presentations made in 1992 and 1993. A published version can be found in *EPODD — Electronic Publishing*, vol. 6(3), September 1993, pp. 283–288.]]

Fernand Baudin, Note de lecture/Book Review : *The Manuel Typographique of Pierre-Simon Fournier le jeune*, facsimile of English trans. by Harry Carter, 1930. 3 volumes. Darmstadt, 1995. ISBN 3-88607-094-8. ; pp. 71–73

[[No abstract or resume for reviews.] The 1930 edition of 3 volumes comprised the two volumes of the *Manuel typographiques* (1764 and 1766), along with Carter's English-language addition, entitled *Fournier on Typefounding*. This last brings out the quality and quantity of Fournier's activities, not only in actual type production but also in his correspondence, in setting standards, in managing his business, in writing articles — a prodigious legacy. The 1995 facsimile edition has an introduction and notes by James Mosley, which again enhance Carter's work, as well as bringing Fournier's work again to the fore to a new audience.]

[Compiled by Christina Thiele]

Articles from *Cahiers* issues can be found in PostScript format at the following site:
`http://www.univ-rennes1.fr/pub/GUTenberg/publicationsPS`

LaTeX

A proposal for citation commands in LaTeX3

Pedro J. Aphalo

1 Introduction

There are several recent published and unpublished papers on the problem of what kinds of citations and bibliographies are commonly used in different disciplines (Rhead, 1990, 1993, and unpublished, Wonneberger and Mittelbach, 1991), one proposal of how LaTeX3 might support them (Rhead, 1991, and unpublished), and one implementation called Camel (Bennett, 1996) more in line with the current proposal than with Rhead's.

Rhead describes three citation schemes: citation-by-key,[1] author-date and short-form. Although he accepts that if possible using a single set of commands for the three citation schemes would be preferable, he argues that this is not possible (Rhead, 1991).

I agree in general with David Rhead's description of the different citation schemes, but in contrast to what he assumes in his proposal for citation commands, I think that it is possible to mark the input for the three citation schemes using the same set of commands. Below I describe an alternative proposal for the syntax of citation commands in LaTeX3 which does not rely on different citation commands for different citation schemes. I hope that having two contrasting proposals available will highlight the tradeoffs involved, and help the developers of LaTeX3 design a good user interface for citations. However, I do not expect either proposal to be adopted without substantial changes.

As a user, I strongly object to having three different sets of commands for citations. (In disciplines like biology different schemes are used by different journals and publishing houses, so having to change from one scheme to a different one is a real problem.) Having a user interface with more commands also makes it more difficult for the user to learn how to use them! I do not object so much to having different `\bibitem`[2] commands because I use BibTeX for all

[1] I prefer the more general name 'citation-by-key' to 'reference-by-number' as used by Rhead, because although the key is usually a number this is not always the case as in `alpha.bst`.

[2] To keep this discussion simple I will assume a system which follows the same general design for the generation of citations and lists of references as that implemented in the current versions of LaTeX and BibTeX. This implementation

my manuscripts, but I think that a consistent syntax would also be highly desirable.

Preserving the separation of contents and format would provide for a generic markup of manuscripts that could be easily translated to in-house formats. The normal LaTeX document styles and bibliography styles could use exactly the same syntax as in-house styles and make electronic submission for publication much easier than nowadays — e.g. Elsevier Science Publishers and Kluwer Academic Publishers have each one a single, but different, generic format for submission of manuscripts as LaTeX source files to *many* of their journals (Elsevier Science, 1995; Kluwer Academic Publishers, 1997). It would be much better if different publishing houses used the same standard format.

Another advantage of abstracting formatting issues into style files is that the same document can be formatted differently for different purposes (e.g. TeXinfo). If one considers the possibility of using LaTeX for documents to be printed but also viewed on-line, using different formats on paper and screens, the use of a single and consistent syntax becomes very important. For the development of LaTeX viewers, such as TechExplorer from IBM (see `http://www.ics.raleigh.ibm.com/ics/techexp.htm`) the use of such a consistent syntax would guarantee their compatibility with many LaTeX source files without any need of editing.

These are some examples of the advantages of generic markup, which are behind the main objective of this proposal: to achieve a generic syntax for citations capable of supporting the different citation schemes.

2 Rationale

Citation-by-key is the only scheme currently supported by LaTeX without extensions. Consequently the available citation commands are too limited and citation styles that add new commands have proliferated (e.g. `chicago.sty, harvard.sty, authordate.sty`). This is not good, and LaTeX3 should aim at providing a complete set of commands flexible enough to provide to the needs of different citation schemes (however, it should provide only a few basic examples of their use in citation styles and an interface for easily defining new citation styles).

The information needed in different citation schemes is not very different and could be thought of as different subsets of a citation's 'full' information set. The amount of information required for this superset is certainly finite and in practice the different subsets overlap a lot. Abstracting a superset common to all three citation schemes is not trivial, but my attempt at doing such an abstraction is the basis for the present proposal. Another requirement is to have this information parsed into small enough units (i.e. having enough 'fields' in the `\bibitem` commands) so that the right pieces can be chosen by the different citation styles. This implies a trade-off between bibliographic data entry against ease of document markup, but as bibliographic data entry can be automated by use of a program such as BibTeX I think that easy of document markup should be favoured.

In practice, provided that some discipline is used when typing a piece of text using the citation commands proposed here (e.g. use of `\citeasnoun` and `\citenoname` even for the citation-by-key scheme), no adjustment would be required in most cases when changing a document from one citation scheme to another. Consequently, there is no reason from the user's point of view that justifies breaking one of the design principles of LaTeX: logic structure and format should be kept separate.

It would be possible either to have in a file only the subset of the bibliographic information needed for the scheme in use, and to regenerate (probably by means of a program like BibTeX) the `\bibitem` commands when switching to a different scheme, or to have the full set of information always available in the `\bibitem` commands. I think that the second option should be favoured because it makes switching between styles a lot easier and also because one may need to include the name of authors or dates of publication in the text independently of the citation scheme being used. Another reason is that the use of the same syntax for `\bibitem` commands for all the schemes (a simpler user interface) would simplify the implementation of mixed citation schemes.

A reason given for having different commands is that some documents use more than one citation scheme. However, the use of this 'mixed' scheme is not a common situation, and its full support should be addressed by special styles and not by allowing the simultaneous use of more than one citation style with LaTeX's default commands. Styles supporting the 'mixed' scheme could rely on optional arguments of the same standard `\cite` commands to switch between citation schemes, and in this way they could remain compatible with the three simple schemes.

Mittelbach and Rowley (1993, p. 2) state that 'It [LaTeX] was designed to separate content and form as much as possible...', and that one of the

could change in the future, but discussing such possible changes is not an aim of this article.

aims for LaTeX3 is to separate the interfaces used for generic markup by the author of a LaTeX document and the specification of how the document elements will be formatted (Mittelbach and Rowley, 1993, p. 5).

Keeping the command set consistent and fully implemented in all styles is the basis for keeping format and structure separate. Such a command set *allows* generic markup which makes it possible to change the citation style without having to edit the whole document to replace incompatible variations of the `\cite` commands.

3 Commands

I propose the following basic set of commands, to be implemented in *all* citation styles. There is a trade-off: the number of commands is larger than what would be needed to support a single citation scheme, especially the citation-by-key scheme, but this is the price that has to be paid for having a single set of commands which can support well all three citation schemes.

I have tried to keep the syntax of the commands consistent with the rest of LaTeX. The biggest departure from 'normal' LaTeX syntax is the use of optional arguments *within* the curly brackets of the main argument of `\cite`.[3] This seems to me the most logical way of making clear that the optional string arguments remain attached to the citation specified by each citation key (the order of citations within a pair of *citation brackets* is *not* guaranteed to be the same as the order of the citation keys supplied as argument: styles may arrange them either in alphabetical or date order, for the author-date scheme, or in key order for the citation-by-key scheme, or leave them in the argument's order). The syntax also assumes that styles that format first citations differently from later ones automatically detect which ones are first citations (this is possible to achieve, and has been implemented by Peter Williams in the Harvard family of bibliography styles). The commands marked • cannot be replaced with simpler ones, those marked ◦ can be replaced with more basic ones but are included because they are used frequently. The commands marked + should be considered optional: a syntax to be used if implemented. `\authorof` and `\yearof` are not citation commands, but are very useful as they guarantee consistency of spelling for author names and consistency for dates. They could also be very useful for generating documents using templates or 'boiler plates'.

[3] A syntax first proposed by David Rhead.

- • `\cite[`*opt*`]{[`*str*`]`*key*`[`*str*`],...`[4]`}`, where *opt* is a style specific option, *str* is a text string, and *key* is the citation key of a `\bibitem` (or of a BibTeX database entry), generates a citation string, including enclosing brackets or footnote(s). Options, if not supported, should be ignored,[5] with a warning except for `f` and `l` below which should be supported whenever they are meaningful and quietly ignored otherwise.
  `\cite[f]{[`*str*`]`*key*`[`*str*`],...}` ... treat as first citation(s) of the *key*(s) even if they are later instances of the citations.
  `\cite[l]{[`*str*`]`*key*`[`*str*`],...}` ... treat as later citation(s) of the *key*(s) even if they are first instances of the citations. (The options `f` and `l` apply to the whole compound citation, because forcing a special format is related to the context of the citation and not to individual *key*s. Special formatting is needed when some citations are not considered to be part of the sequence of citations in the main body of a document, as is frequently the case for citations within tables. Moreover, the `\cite` commands with `f` or `l` options should be ignored when automatically deciding whether a citation to a given *key* is the first or a later one.)
- ◦ `\citeaffixed[`*opt*`]{`*aff*`}{[`*str*`]`*key*`[`*str*`],...}` generates a citation string with *aff* affixed, including enclosing brackets or footnote(s). `f` and `l`, and other options as in `\cite`.
- ◦ `\citeasnoun[`*opt*`]{`*key*`[`*str*`]}` generates a string with author's name(s) and citation to be used as a noun in a sentence. `f` and `l`, and other options as in `\cite` above. (Rhead would not include this command and the next one, arguing that the authors' names are not part of the citation but rather part of the text. I think that he is only partly right, because at least in the author-date scheme, whether the name list is abbreviated or not depends on whether the names precede the first or a later instance of a citation. In other words, the instructions to authors of several biological journals do implicitly consider them part of the citation.)
- ◦ `\citepossesive[`*opt*`]{`*key*`[`*str*`]}` generates a string with author's name(s) and citation to be

[4] The sequence of *key* plus attached strings can be repeated, using comma as separator. This also applies to the next command, but not to other cite commands described below.

[5] This allows compatibility with other citation styles that do not need the extra information. However, style authors should strive to give unique names to options, unless they have the same function.

used as a possessive in a sentence. `f` and `l`, and other options as in `\cite` above.

- `\citenoname[`*opt*`]{[`*str*`]`*key*`[`*str*`]}` behaves as `\cite` but does not include the authors in the author-date scheme. `f` and `l`, and other options as in `\cite` above. (It is needed for some 'MLA' and 'MHRA' examples in one of Rhead's unpublished papers. Rhead uses the name `\dcite` for this command in his proposal for supporting the author-date scheme.)
- `\citation{`*str*`}` formats the string(s) as a single compound citation — i.e. encloses them in brackets or sets them as a footnote —, and simultaneously disables the generation of additional brackets or footnotes by `\cite` commands used as part of its *str* argument. (The logical markup has a different meaning than a `\cite` with multiple keys, although in some cases the formatting may be the same: the list of keys in a `\cite` command is unordered and the style is allowed to sort them. In contrast, the different `\cite` commands within a citation are ordered, and any connecting text also remains always where it is in the input text, between a given pair of `\cite` commands. This command and the following one make the `\citeNP` family of commands used in `chicago.sty` and `ltugboat.cls` with the `harvardcite` option, as used in the source of this document, redundant.)
- `\nocitation{`*str*`}` does not format the string(s) as a citation — i.e. does not enclose them in brackets or set them as a footnote —, but similarly to `\citation` disables the generation of additional brackets or footnotes by `\cite` commands used as part of its *str* argument. (To be used for example in tables summarizing data from other publications, in which table cells normally contain citations formatted as normal text, without brackets in the case of the author-date scheme.)
- `\nocite{`*key*`,...}` generates no output, but forces inclusion of references to the *key*(s) in the list of references.

+ `\bibref{`*key*`}` generates a full reference at the position in the document where the command appears, not a citation. (To be used in abstracts in which references are usually given in full rather than as a pointer to a list of references, and also for writing commented lists of suggested reading.)
+ `\yearof{`*key*`}` year of *key*; it is not a citation — i.e. a reference to *key* is not included in the list of references.
+ `\authorof[`*opt*`]{`*key*`}` author or authors of *key*; it is not a citation — i.e. a reference to *key* is not included in list of references. `f` and `l`, and other options as in `cite` above.

4 Related problems

4.1 Pinpointing locations within a reference

In the previous section, to keep the description of the `\cite` commands simple, I have ignored the problem of citing specific parts of a publication. The syntax for `\cite` given above does not explicitly support style and language independent pinpointing to pages, sections, chapters, etc. Full support for pinpoints should not only provide the pinpoint prefix and typeface, but also scan their argument to determine whether a plural is needed (e.g. 'p.' or 'pp.' for pages). Styles could differ also in the location of the pinpoint: (Hoff 1992, pp. 143–179) vs. (pages 143–179 of Hoff 1992).

This functionality could be accommodated by the following syntax for the `\cite` command: `\cite{Hoff92<p:143--179>}`; also valid input is `\cite{Hoff92<p:257>[,␣describes␣well␣my␣feelings]}`. In other words `<...>` would be used for pinpoints, and `[...]` for strings. The problem of this approach is that it probably would make the code for `\cite` complicated. The revised syntax of the `\cite` command would become:

- `\cite[`*opt*`]{[`*str*`]`*key*`<`*pinpoint*`>[`*str*`],...}` with arguments enclosed in `[ ]` and `< >` being optional, or
- `\cite[`*opt*`]{`*key*`<`*pinpoint*`>,...}` if the optional *str* arguments are considered redundant.

with a similar syntax for all other `\cite` commands.

For either of these two last variants of the syntax for `\cite` commands the proposed pinpoint arguments are: `vol:`*volume(s)*, `part:`*part(s)*, `ch:`*chapter(s)*, `sec:`*section(s)*, `p:`*page(s)*, `fig:`*figure(s)*, `tab:`*table(s)*, `plate:`*plate(s)*, `eq:`*equation(s)*, `th:`*theorem(s)*, `col:`*column(s)*, `para:`*paragraph(s)*, `line:`*line(s)*.
Compound pinpoints such as `<vol:3,ch:9,p:1012>` are also valid.

4.2 Signals and other terms

What Rhead calls "signals" are very often used in texts about law, and less frequently in other disciplines. For these terms, the typefaces and abbreviations used depend on house styles. An optional style could define them, but it is arguable whether they differ from the more general problem of using abbreviations and symbols, except for the fact that

they are widely used. The advantage of providing an optional style file with their default definitions as part of LaTeX3 would be the standardisation of the names used for this group of commands. Rhead proposes a list of such commands as used in texts about law: `\accord`, `\Accord`, `\and`, `\butcf`, `\Butcf`, `\butsee`, `\Butsee`, `\cf`, `\Cf`, `\compare`, `\Compare`, `\contra`, `\Contra`, `\eg`, `\Eg`, `\etseq`, `\ibid`, `\Ibid`, `\id`, `\Id`, `\infra`, `\loccit`, `\opcit`, `\see`, `\See`, `\seealso`, `\Seealso`, `\seegenerally`, `\Seegenerally`, `\supra`, `\re`, `\Re`, `\versus`, `\with`. Only some of them are used in texts unrelated to law; the full list should be implemented only in law-specific styles. Some of these "signals" can be used together with citations (a) as explanatory text: `\citeaffixed{\see␣}{Hoff92}` would print as '(see Hoff 1992)', or (b) as pinpoints: `\cite{Hoff92[,␣\loccit]}` would print as '(Hoff 1992, *loc. cit.*)'. I think that pinpoints like *loc. cit.* and *op. cit.* should be handled automatically by citation styles because they are a formatting issue and have no intrinsic meaning — e.g. (Hoff, *op. cit.*) in the right context has exactly the same meaning as (Hoff 1992). In contrast *cf.*, *see*, *versus*, etc. should be specified by the author because they alter the meaning of citations.

5 Examples

A few simple examples of the use of these commands and of how the output might look for the different citation schemes are provided below in the following order: (i) author-date, (ii) citation-by-key using numeric keys, (iii) citation-by-key using alphanumeric keys, and (iv) short-form. In the cases in which the citations within a single `\cite` command could be automatically sorted in either alphabetical or chronological order, both possibilities are shown. For the short-form scheme fake footnotes are given at the end of each item in the list of examples, the numbers for numeric keys in the examples are also faked, but not the authors and titles.

▷ `\cite{Borges78,Hudson18}`
(Borges, 1978; Hudson, 1918) or (Hudson, 1918; Borges, 1978)
(1,2)
[Bor78, Hud18] or [Hud18, Bor78]
[1,2]

[1] Borges, J. L., *El Libro de los Seres Imaginarios.*
[2] Hudson, W. H., *Far Away and Long Ago.*

▷ `\citeaffixed{see}{Borges78,Hudson18}`
(see Borges, 1978; Hudson, 1918) or (see Hudson, 1918; Borges, 1978)
(see 1,2) or even (1,2)
[see Bor78, Hud18] or [see Hud18, Bor78]
[1,2]

[1] See Borges, J. L., *El Libro de los Seres Imaginarios.*
[2] See Hudson, W. H., *Far Away and Long Ago.*

▷ `\citeasnoun{Borges78}`
Borges (1978)
Borges (1)
Borges [Bor78]
Borges[1]

[1] Borges, J. L., *El Libro de los Seres Imaginarios.*

▷ `\citepossesive{Borges78}`
Borges' (1978)
Borges' (1)
Borges' [Bor78]
Borges'[1]

[1] Borges, J. L., *El Libro de los Seres Imaginarios.*

▷ `\citenoname{Borges78}`
(1978)
(1)
[Bor78]
[1]

[1] Borges, J. L., *El Libro de los Seres Imaginarios.*

▷ `\cite{Borges78<p:45-46>, Hudson18<ch:3>}`
(Hudson, 1918, chapter 3; Borges, 1978, pp. 45–46) or (Borges, 1978, pp. 45–46; Hudson, 1918, chapter 3)
(1, pp. 45–46, 2 chapter 3) or (1 chapter 3, 2 pp. 45–46)
[Bor78 pp. 45–46, Hud18 chapter 3] or [Hud18 chapter 3, Bor78 pp. 45–46]
[1,2]

[1] Borges, J. L., *El Libro de los Seres Imaginarios*, pp. 45–46.
[2] Hudson, W. H., *Far Away and Long Ago*, chapter 3.

▷ `\citation{see␣\cite{Borges78}␣or \cite{Hudson18}}`
(see Borges, 1978 or Hudson, 1918)
(see 1 or 2) or (see 2 or 1), where the first number always refers to `Borges78`
[see Bor78 or Hud18]
[1]

[1] See Borges, J. L., *El Libro de los Seres Imaginarios* or Hudson, W. H., *Far Away and Long Ago.*

6 Caveats

The command set proposed cannot support the needs of all legal texts. References to cases and tables of cases require special commands, but citations of books, articles in periodicals, etc., can most probably be supported by the commands given above. In one of his unpublished manuscripts, Rhead gives an example of how the necessary extensions could be supported by a law-specific style. A basic example could be provided as part of LaTeX3 so as to provide a guideline for programmers of styles for legal texts.

While working on this proposal I have deliberately 'forgotten' all problems that have to do with the implementation of the commands. My philosophy is that first we should have clear what we want, and only afterwards worry about the implementation. Only as a last resort should we change the syntax to suit the limitations of TeX. My idea is that we should be very open minded about implementation issues, and even consider *heretical* alternatives such as the use of a preprocessor that reads not only the `.aux` file but also the `.tex` file or maybe even generates the file to be processed by TeX replacing the `\cite` commands with something else that is easier for TeX to process.[6] However, we should not forget that using preprocessors has serious drawbacks: (1) the preprocessors should be as portable as TeX itself, and ports should exist for all platforms and operating systems for which TeX is available. (2) preprocessing or additional passes through TeX itself slow down document formatting.

7 Acknowledgements

This paper is a revised version of my contribution to task 04 of the volunteer work for the LaTeX3 project (Mittelbach, Rowley, and Downes, 1993). This proposal owes much to the extensive work of David Rhead (mostly unpublished, but kindly made available by him) and to the authors of the many citation styles that are available in the CTAN archives. David Rhead (UK), Peter Williams (Australia), John Grace (UK), John Wells (Canada) and Mario Natiello (Sweden) kindly answered my questions and made useful suggestions. A suggestion of Frank Bennett (UK) plus his efforts in implementing Camel were the main incentive behind the revision of this manuscript for *TUGboat*. Finally, the detailed comments from an anonymous referee helped me improve this document.

[6] As long as the syntax of the commands remains consistent, support for different schemes could even rely on different preprocessors!

References

Bennett, F. G., Jr. "Camel: kicking over the bibliographic traces in BibTeX". *TUGboat* **17**(1), 22–28, 1996.

Borges, J. L. *El Libro de los Seres Imaginarios*. Emecé Editores, Buenos Aires, 1978. This edition first published in 1967 by Kier, Buenos Aires.

Elsevier Science. "Preparing Articles with LaTeX. Instructions to Authors for Preparing Compuscripts". 1995. This is file `instraut.dvi`, included in the ESP-LaTeX package archived in the CTAN archives in the directory `macros/latex/contrib/supported/elsevier`.

Hoff, B. *The Te of Piglet*. Dutton, Penguin Books, New York, 1992.

Hudson, W. H. *Far Away and Long Ago*. J. M. Dent and Sons, London, 1918. Reprinted 1982, Eland Books, London.

Kluwer Academic Publishers. "User manual for `kluwer.cls`, version 1997/05/30. Instructions for authors". 1997. This is file `usrman.tex`, included with the LaTeX class archived in the CTAN archives in the directory `macros/latex/contrib/supported/kluwer`.

Mittelbach, F. and C. Rowley. "The LaTeX3 Project". Public, official document, LaTeX3 Project, London, 1993. File `l3d001.tex` archived in CTAN.

Mittelbach, F., C. Rowley, and M. Downes. "Volunteer work for the LaTeX3 project (Version 6.2a)". Public document, LaTeX3 Project, 1993. File `vol-task.tex` archived in CTAN.

Rhead, D. "Towards BibTeX style-files that implement principal standards". *TeXline* (10), 2–8, 1990.

Rhead, D. "How might LaTeX 3.0 deal with citations and reference lists?". *TeXline* (13), 13–20, 1991.

Rhead, D. "The "operational requirement" for support of bibliographic references by LaTeX 3". *TUGboat* **14**(4), 425–433, 1993.

Wonneberger, R. and F. Mittelbach. "BibTeX Reconsidered". *TUGboat* **12**(1), 111–124, 1991. TeX 90 Conference Proceedings.

◇ Pedro J. Aphalo
Faculty of Forestry
University of Joensuu
P.O. Box 111
FIN-80101 Joensuu
Finland
`pedro.aphalo@joensuu.fi`
`http://cc.joensuu.fi/~aphalo/`

The LaTeX3 Programming Language—a proposed system for TeX macro programming

David Carlisle, Chris Rowley and Frank Mittelbach

Abstract

This paper gives s brief introduction to a new set of programming conventions that have been designed to meet the requirements of implementing large scale TeX macro programming projects such as LaTeX.

The main features of the system described are:

- classification of the macros (or, in LaTeX terminology, commands) into LaTeX functions and LaTeX parameters, and also into modules containing related commands;
- a systematic naming scheme based on these classifications;
- a simple mechanism for controlling the expansion of a function's arguments.

A system such as this is being used experimentally as the basis for TeX programming within the LaTeX3 project. Note that the language is not intended for either document mark-up or style specification.

This paper is based on a talk given by David Carlisle in San Francisco, July 1997, but it describes the work of several people: principally Frank Mittelbach and Denys Duchier, together with Johannes Braams, David Carlisle, Michael Downes, Alan Jeffrey, Chris Rowley and Rainer Schöpf.

1 Introduction

This paper describes the conventions for a TeX-based programming language which is intended to provide a more consistent and rational environment for the construction of large scale systems, such as LaTeX, using TeX macros.

Variants of this language have been in use by The LaTeX3 Project Team since around 1990 but the syntax specification to be outlined here should *not* be considered final. This is an experimental language thus many aspects, such as the syntax conventions and naming schemes, may (and probably will) change as more experience is gained with using the language in practice.

The next section shows where this language fits into a complete TeX-based document processing system. We then describe the major features of the syntactic structure of command names, including the argument specification syntax used in function names.

The practical ideas behind this argument syntax will be explained, together with the semantics of the expansion control mechanism and the interface used to define variant forms of functions. The paper also discusses some advantages of the syntax for parameter names.

As we shall demonstrate, the use of a structured naming scheme and of variant forms for functions greatly improves the readability of the code and hence also its reliability. Moreover, experience has shown that the longer command names which result from the new syntax do not make the process of *writing* code significantly harder (especially when using a reasonably intelligent editor).

The final section gives some details of our plans to distribute parts of this system during the next year. More general information concerning the work of the LaTeX3 Project can be found in [4].

2 Languages and interfaces

It is possible to identify several distinct languages related to the various interfaces that are needed in a TeX-based document processing system. This section looks at those we consider most important for the LaTeX3 system.

Document mark-up This comprises those commands (often called tags) that are to embedded in the document (the `.tex` file).

It is generally accepted that such mark-up should be essentially *declarative*. It may be traditional TeX-based mark-up such as LaTeX 2_ε, as described in [3] and [2], or SGML-based mark-up such as XML.

One problem with more traditional TeX coding conventions (as described in [1] is that the names and syntax of TeX's primitive formatting commands are ingeniously designed to be 'natural' when used directly by the author as document mark-up or in macros. Ironically, the ubiquity (and widely recognised superiority) of logical mark-up has meant that such explicit formatting commands are almost never needed in documents or in author-defined macros. Thus they are used almost exclusively by TeX programmers to define higher-level commands; and their idiosyncratic syntax is not at all popular with this community. Moreover, many of them have names that could be very useful as document mark-up tags were they not pre-empted as primitives (e.g., `\box` or `\special`).

Designer interface This relates a (human) typographic designer's specification for a document to a program that 'formats the document'. It should ideally use a declarative language that facilitates expression of the relationship and spacing rules specified for the layout of the various document elements.

This language is not embedded in document text and it will be very different in form to the document mark-up language. For SGML-based systems the DSSSL language may come to play this role. For LaTeX, this level was almost completely missing from LaTeX2.09; LaTeX 2_ε made some improvements in this area but it is still the case that implementing a design specification in LaTeX requires far more 'low-level' coding than is acceptable.

Programmer interface This language is the implementation language in which the basic typesetting functionality is implemented, building upon the primitives of TeX (or a successor program). It may also be used to implement the previous two languages 'within' TeX, as in the current LaTeX system.

Only the last of these three interfaces is covered by this paper, which describes a system aimed at providing a suitable basis for coding large scale projects in TeX (but this should not preclude its use for smaller projects). Its main distinguishing features are summarised here.

- A consistent naming scheme for all commands, including TeX primitives.
- The classification of commands as LaTeX functions or LaTeX parameters, and also their division into modules according to their functionality.
- A simple mechanism for controlling argument expansion.
- Provision of a set of core LaTeX functions that is sufficient for handling programming constructs such as queues, sets, stacks, property lists.
- A TeX programming environment in which, for example, all white space is ignored.

3 The naming scheme

The naming conventions for this programming language distinguish between *functions* and *parameters*. Functions can have arguments and they are executed. Parameters can be assigned values and they are used in arguments to functions; they are not directly executed but are manipulated by mutator and accessor functions. Functions and parameters with a related functionality (for example accessing counters, or manipulating token-lists, etc.) are collected together into a *module*.

Note that all these terms are only LaTeX terminology and are not, for example, intended to indicate that the commands have these properties when considered in the context of basic TeX or in any more general programming context.

3.1 Examples

Before giving the details of the naming scheme, here are a few typical examples to indicate the flavour of the scheme; first some parameter names.

`\l_tmpa_box` is a local parameter (hence the `l_` prefix) corresponding to a box register.
`\g_tmpa_int` is a global parameter (hence the `g_` prefix) corresponding to an integer register (i.e., a TeX count register).
`\c_empty_toks` is the constant (`c_`) token register parameter that is for ever empty.

Now here is an example of a typical function name.

`\seq_push:Nn` is the function which puts the token list specified by its second argument onto the stack specified by its first argument. The different natures of the two arguments are indicated by the `:Nn` suffix. The first argument must be a single token which 'names' the stack parameter: such single-token arguments are denoted `N`. The second argument is a normal TeX 'undelimited argument', which may either be a single token or a balanced, brace-delimited token list (which we shall here call a *braced token list*): the `n` denotes such a 'normal' argument form.

`\seq_push:cn` would be similar to the above, but in this case the `c` means that the stack-name is specified in the first argument by a token list that expands, using `\csname...`, to a control sequence that is the *name* of the stack parameter.
The names of these two functions also indicate that they are in the module called `seq`.

3.2 Formal syntax of the conventions

We shall now look in more detail at the syntax of these names.
The syntax of parameter names is as follows:

\⟨*access*⟩_⟨*module*⟩_⟨*description*⟩_⟨*type*⟩

The syntax of function names is as follows:

\⟨*module*⟩_⟨*description*⟩:⟨*arg-spec*⟩

3.3 Modules and descriptions

The syntax of all names contains

⟨*module*⟩ and ⟨*description*⟩:

these both give information about the command.

A *module* is a collection of closely related functions and parameters. Typical module names include `int` for integer parameters and related functions, `seq` for sequences and `box` for boxes.

Packages providing new programming functionality will add new modules as needed; the programmer can choose any unused name, consisting of letters only, for a module.

The *description* gives more detailed information about the function or parameter, and provides a unique name for it. It should consist of letters and, possibly, _ characters.

3.4 Parameters: access and type

The ⟨*access*⟩ part of the name describes how the parameter can be accessed. Parameters are primarily classified as local, global or constant (there are further, more technical, classes). This *access* type appears as a code at the beginning of the name; the codes used include:

c constants (global parameters whose value should not be changed);

g parameters whose value should only be set globally;

l parameters whose value should only be set locally.

The ⟨*type*⟩ will normally (except when introducing a new data-type) be in the list of available *data-types*; these include the primitive TEX data-types, such as the various registers, but to these will be added data-types built within the LATEX programming system.

Here are some typical data-type names:

int integer-valued count register;

toks token register;

box box register;

fint 'Fake-integer': (or fake-counter) a data type created to avoid problems with the limited number of available count registers in (standard) TEX;

seq 'sequence': a data-type used to implement lists (with access at both ends) and stacks;

plist property list

When the ⟨*type*⟩ and ⟨*module*⟩ are identical (as often happens in the more basic modules) the ⟨*module*⟩ part is often omitted for aesthetic reasons.

3.5 Functions: argument specifications

Function names end with an ⟨*arg-spec*⟩ after a colon. This gives an indication of the types of argument that a function takes, and provides a convenient method of naming similar functions that differ only in their argument forms (see the next section for examples).

The ⟨*arg-spec*⟩ consists of a (possibly empty) list of characters, each denoting one argument of the function. It is important to understand that 'argument' here refers to the effective argument of the LATEX function, not to an argument at the TEX-level. Indeed, the top level TEX macro that has this name typically has no arguments. This is an extension of the existing LATEX convention where one says that `\section` has an optional argument and a mandatory argument, whereas the TEX macro `\section` actually has zero parameters at the TEX level, it merely calls an internal LATEX command which in turn calls others that look ahead for star forms and optional arguments.

The list of possible argument specifiers includes the following.

n Unexpanded token or braced token list.
This is a standard TEX undelimited macro argument.

o One-level-expanded token or braced token list.
This means that the argument is expanded one level, as is done by `\expandafter`, and the expansion is passed to the function as a braced token list. Note that if the original argument is a braced token list then only the first token in that list is expanded.

x Fully-expanded token or braced token list.
This means that the argument is expanded as in the replacement text of an `\edef`, and the expansion is passed to the function as a braced token list.

c Character string used as a command name.
The argument (a token or braced token list) must, when fully expanded, produce a sequence of characters which is then used to construct a command name (via `\csname`, `\endcsname`). This command name is the single token that is passed to the function as the argument.

N Single token (unlike `n`, the argument must *not* be surrounded by braces).
A typical example of a command taking an `N` argument is `\def`, in which the command being defined must be unbraced.

O One-level-expanded single token (unbraced).
As for `o`, the one-level expansion is passed (as a braced token list) to the function.

X Fully-expanded single token (unbraced).
As for `x`, the full expansion is passed (as a braced token list) to the function.

C Character string used as a command name then one-level expanded.
The form of the argument is exactly as for `c`, but the resulting token is then expanded one level (as for `O`), and the expansion is passed to the function as a braced token list.

p Primitive TEX parameter specification.
This can be something simple like `#1#2#3`, but may use arbitrary delimited argument syntax such as: `#1,#2\q_stop#3`.

T,F These are special cases of `n` arguments, used for the true and false code in conditional commands.

There are two other specifiers with more general meanings:

D This means: **Do not use**. This special case is used for TEX primitives and other commands that are provided for use only while bootstrapping the LATEX kernel. If the TEX primitive needs to be used in other contexts it will be given an alternative, more appropriate, name with a useful argument specification. The argument syntax of these is often weird, in the sense described next.

w This means that the argument syntax is 'weird' in that it does not follow any standard rule. It is used for functions with arguments that take non standard forms: examples are TEX-level delimited arguments and the boolean tests needed after certain primitive `\if...` commands.

4 Expansion control

4.1 Simpler means better

Anyone who programs in TEX is frustratingly familiar with the problem of arranging that arguments to functions are suitably expanded before the function is called. To illustrate how expansion control can bring instant relief to this problem we shall consider two examples copied from `latex.ltx`.

```
\global
\expandafter
  \expandafter
\expandafter
    \let
\expandafter
  \reserved@a
\csname \curr@fontshape \endcsname
```

This first piece of code is in essence simply a global `\let`. However, the token to be defined is obtained by expanding `\reserved@a` one level; and, worse, the token to which it is to be let is obtained by fully expanding `\curr@fontshape` and then using the characters produced by that expansion to construct a command name. The result is a mess of interwoven `\expandafter` and `\csname` beloved of all TEX programmers, and the code is essentially unreadable.

Using the conventions and functionality outlined here, the task would be achieved with code such as this:

```
\glet:Oc \g_reserved_a_tlp
           \l_current_font_shape_tlp
```

The command `\glet:Oc` is a global `\let` that expands its first argument once, and generates a command name out of its second argument, before making the definition. This produces code that is far more readable and more likely to be correct first time.

Here is the second example.

```
\expandafter
  \in@
\csname sym#3%
  \expandafter
    \endcsname
  \expandafter
    {%
  \group@list}%
```

This piece of code is part of the definition of another function. It first produces two things: a token list, by expanding `\group@list` once; and a token whose name comes from '`sym#3`'. Then the function `\in@` is called and this tests if its first argument occurs in the token list of its second argument.

Again we can improve enormously on the code. First we shall rename the function `\in@` according to our conventions. A function such as this but taking two normal '`n`' arguments might reasonably be named `\seq_test_in:nn`; thus the variant function we need will be defined with the appropriate argument types and its name will be `\seq_test_in:cO`. Now this code fragment will be simply:

```
\seq_test_in:cO {sym#3} \l_group_seq
```

Note that, in addition to the lack of `\expandafter`, the space after the `}` will be silently ignored since all white space is ignored in this programming environment.

4.2 New functions from old

For many common functions the LATEX3 kernel will provide variants with a range of argument forms, and similarly it is expected that extension packages providing new functions will make them available in the all the commonly needed forms.

However, there will be occasions where it is necessary to construct a new such variant form; therefore the expansion module provides a straightforward mechanism for the creation of functions with any required argument type, starting from a function that takes 'normal' TEX undelimited arguments.

To illustrate this let us suppose you have a 'base function' `\demo_cmd:nnn` that takes three normal arguments, and that you need to construct the variant `\demo_cmd:cnx`, for which the first argument is used to construct the *name* of a command, whilst the third argument must be fully expanded before being passed to `\demo_cmd:nnn`. To produce the variant form from the base form, simply use this:

```
\exp_def_form:nnn {demo_cmd} {nnn} {cnx}
```

This defines the variant form so that you can then write, for example:

```
\demo_cmd:cnx {abc} {pq} {\rst \xyz }
```

rather than ... well, something like this!

```
\def \tempa {{pq}}%
\edef \tempb {\rst \xyz}%
\expandafter
  \demo@cmd
\csname abc%
  \expandafter
    \expandafter
  \expandafter
      \endcsname
  \expandafter
    \tempa
  \expandafter
    {%
  \tempb
    }%
```

As a further example, you may wish to declare a function `\demo_cmd_b:xcxcx`, as a variant of an existing function `\demo_cmd_b:nnnnn`, that fully expands arguments 1, 3 and 5, and produces commands to pass as arguments 2 and 4 using `\csname`. The definition you need is simply

```
\exp_def_form:nnn
  {demo_cmd_b} {nnnnn} {xcxcx}
```

This extension mechanism is written so that if the same new form of some existing command is implemented by two extension packages then the two definitions will be identical and thus no conflict will occur.

5 Parameter assignments and accessor functions

5.1 Checking assignments

One of the advantages of having a consistent scheme is that the system can provide more extensive error-checking and debugging facilities. For example, an accessor function that makes a *global* assignment of a value to a parameter can check that it is not passed the name of a *local* parameter as that argument: it does this by checking that the name starts with `\g_`.

Such checking is probably too slow for normal use, but the code can have hooks built in that allow a format to be made in which all functions perform this kind of check.

A typical section of the source[1] for such code might look like this (recall that all white space is ignored):

```
%<*!check>
\let_new:NN
  \toks_gset:Nn \tex_global:D
%</!check>
%<*check>
\def_new:Npn
  \toks_gset:Nn #1
    {
    \chk_global:N #1
    \tex_global:D #1
    }
%</check>
```

In the above code the function `\toks_gset:Nn` takes a single token (`N`) specifying a token register, and globally sets it to the value passed in the second argument.

A typical use of it would be:

```
\toks_gset \g_xxx_toks {<some value>}
```

In the normal definition, `\toks_gset` can be simply `\let` to `\global` because the primitive TEX token register does not require any explicit assignment function: this is done by the `%<*!check>` code above.

The alternative definition first checks that the argument passed as `#1` is the name of a global parameter and raises an error if it is not. It does this by taking apart the command name passed as `#1` and checking that it starts `\g_`.

5.2 Consistency

The primitive TEX syntax for register assignments has a very minimal syntax and, apart from box functions, there are no explicit functions for assigning values to these registers.

This makes it impossible to implement alternative data-types with a syntax that is both consistent and at all similar to the syntax for the primitives; moreover, it encourages a coding style that is very error prone.

As in the `\toks_gset:Nn` example given above, all LATEX data-types are provided with explicit functions for assignment and for use, even when these have essentially empty definitions. This allows for better error-checking as described above; it also allows the construction of further data-types with a similar interface, even when the implementation of the associated functions is very complex.

For example, the 'fake-counter' (`fint`) data-type mentioned above will appear at the LATEX programming level to be exactly like the data-type based on primitive count registers; however, internally it makes no use of count registers. Typical functions in this module are illustrated here.

[1] This code uses the docstrip system described in [2], Section 14.3.

`\fint_new:N \l_tmpa_fint`

This declares the local parameter `\l_example_fint` as a fake-counter.

`\fint_add:Nn \l_example_fint \c_thirty_two`

This increments the value of this fake-counter by 32.

6 The experimental distribution

The initial implementations of a LaTeX programming language using this kind of syntax remain unreleased (and not completely functional); they partly pre-date LaTeX 2$_\varepsilon$! The planned distribution will provide a subset of the functionality of those implementations, in the form of packages to be used on top of LaTeX 2$_\varepsilon$.

The intention is to allow experienced TeX programmers to experiment with the system and to comment on the interface. This means that ***the interface will change***. No part of this system, including the name of anything, should be relied upon as being available in a later release. Please do *experiment* with these packages, but do *not* use them for code that you expect to keep unchanged over a long period.

In view of the intended experimental use for this distribution we shall, in the first instance, produce only a few modules for use with LaTeX 2$_\varepsilon$. These will set up the conventions and the basic functionality of, for example, the expansion mechanism; they will also implement some of the basic programming constructs, such as token-lists and sequences. They are intended only to give a flavour of the code: the full LaTeX3 kernel will provide a very rich set of programming constructs so that packages can efficiently share code, in contrast with the situation in the current LaTeX where every large package must implement its own version of queues, stacks, etc., as necessary.

In the first release of this experimental system at least the following modules will be distributed.

l3names This sets up the basic naming scheme and renames all the TeX primitives. If it is loaded with the option `[removeoldnames]` then the old primitive names such as `\box` become *undefined* and are thus available for user definition. Caution: use of this option will certainly break existing TeX code!

l3basics This contains the basic definition modules used by the other packages.

l3tlp This implements a basic data-type, called a *token-list pointer*, used for storing named token lists: these are essentially TeX macros with no arguments.

l3expan This is the argument expansion module discussed above.

l3quark A 'quark' is a command that is defined to expand to itself! Therefore they must never be expanded as this will generate infinite recursion; they do however have many uses, e.g., as special markers and delimiters within code.

l3seq This implements data-types such as queues and stacks.

l3prop This implements the data-type for 'property lists' that are used, in particular, for storing key/value pairs.

This distribution will also contain the LaTeX source for the latest version of this document, a docstrip install file and two small test files.

In later releases we plan to add further modules and a full-fledged example of the use of the new language: a proto-type implementation for the ideas described in the article 'Language Information in Structured Documents: A Model for Mark-up and Rendering' [5].

References

[1] Donald E Knuth *The TeXbook*. Addison-Wesley, Reading, Massachusetts, 1984.

[2] Goossens, Mittelbach and Samarin. *The LaTeX Companion*. Addison-Wesley, Reading, Massachusetts, 1994.

[3] Leslie Lamport. *LaTeX: A Document Preparation System*. Addison-Wesley, Reading, Massachusetts, second edition, 1994.

[4] Frank Mittelbach and Chris Rowley. The LaTeX3 Project. *TUGboat*, **18**, 195–198, 1997.

[5] Frank Mittelbach and Chris Rowley. Language Information in Structured Documents: A Model for Mark-up and Rendering. *TUGboat*, **18**, 199–205, 1997.

◇ David Carlisle, Chris Rowley and Frank Mittelbach
LaTeX3 project
latex-l@urz.uni-heidelberg.de

A regression test suite for LaTeX 2ε

Frank Mittelbach

Abstract

This paper describes the history of the development of a regression test suite for LaTeX and its importance for the release of stable and reliable future distributions of that software. A more detailed description of the concepts and the implementation of the test suite will be given in [1].

As experience shows that there can't be enough test files in such a suite, we make a plea to the TeX community to help us in making LaTeX distributions even more reliable by joining a new volunteer group working on the task of updating and adding to this suite.

1 Introduction

Back in 1992 when the LaTeX3 team took over maintenance of LaTeX and started to work on the current LaTeX version [3], also known as LaTeX 2ε, I had the idea of producing a test environment for LaTeX that would help us in providing stable and reliable distributions. My idea originated in the `trip` test for the TeX program [2], a fixed test file which is run through TeX, containing code that tries to exercise as many boundary cases as Don Knuth could think of. The output of this run is then compared with a set of files certified by Don to contain the correct information. Only if a new implementation of the TeX program produces the same output (with well defined minor deviations in certain places) is it allowed[1] to be called TeX. The idea behind this is to ensure that TeX behaves identically on all implementations and the `trip` test was the measure proving this.

With LaTeX the idea was not to ensure that it is identical on all platforms — this is automatically the case if the standard installation is obtained and the installation procedures are applied — but to ensure that that new releases of LaTeX do not inadvertently modify the behavior of commands. Since LaTeX is a large and complex system, this is definitely a non-trivial task: in 'fixing' one bug, it is often necessary to modify the definitions of several 'internal' commands, and these may in turn affect many other commands which have no obvious connection with the original problem.

We have had some pretty disastrous experiences of this type, often finding that harmless looking corrections had effects on what seemed, at first glance, completely unrelated areas. This is in part due to the fact that LaTeX is based on the macro language of TeX, which allows reuse and redefinition of arbitrary code fragments.

For that reason we started working on a concept for automated tests to detect such problems. When that system was available, we asked for volunteers to help us in building up a suitable test suite for LaTeX (which at that time was LaTeX 2.09). Part of the rationale behind this work was to ensure that a future transition from LaTeX 2.09 to LaTeX 2ε (for which development was under way) would become as painless as possible, i.e., these tests were also supposed to ensure that the new code for LaTeX 2ε would not change the user interface behavior without detection.

This approach seems to us to have been very successful; this is in large parts due to the quality and quantity of the work of the volunteers helping us at that time, in particular Daniel Flipo and Chris Martin. Figure 1 shows an excerpt from the volunteer task list from 1993 describing this task (and my rather optimistic time requirements for it).

When LaTeX 2ε was released for the first time in 1994 we updated the regression test support macros and tried to improve the test suite by adding new test files when we fixed bugs or when we added new functionality to LaTeX. However, being human, we have not followed this practice as rigorously as we should have: especially since the first releases it has become more and more common for us to fix a small bug without spending the additional time necessary to also write a test file that exhibits the correct behavior.

Today our test suite has about 300 test files which are automatically executed and compared before a new release hits the streets. And indeed, these test files have saved us from embarrassment many times already.

2 This year's boo-boos!

However, results show that such a suite can never be large enough to avoid the need for a patch release once in a while. It is particularly important that new features, such as the release of additional files or the correction of recently found bugs, get tested and frozen within this suite so that there is no unexpected change later on. For example, with the December 1997 release we added the packages `calc`

[1] An additional requirement according to the `trip` test documentation is that the author of the TeX implementation has to be satisfied with the product. In other words, a simple program that throws away all its input and always output the files needed to satisfy the `trip` test would be allowed to call itself TeX as long as the author of that program is happy with it.

Validating LaTeX2.09

Writing test files for regression testing: checking bug fixes and improvements to verify that they don't have undesirable side effects; making sure that bug fixes really correct the problem they were intended to correct; testing interaction with various document styles, style options, and environments.
We would like three kinds of validation files:

1. General documents.
2. Exhaustive tests of special environments/modules such as tables, displayed equations, theorems, floating figures, pictures, etc.
3. Bug files containing tests of all bugs that are supposed to be fixed (as well as those that are not fixed, with comments about their status).

A procedure for processing validation files has been devised; details will be furnished to anyone interested in this task.

Estimated time required: 2 to 3 weeks, could be divided up.

Coordinator [25 August 1992]:
Daniel Flipo `flipo@citil.citilille.fr`
Other volunteers:
Chris Martin `cs1cwm@sunc.sheffield.ac.uk`

Figure 1: An excerpt from the volunteer task list 1993

and `textcomp` to the distribution but, due to time constraints, did not add to the suite additional test files designed to exercise these packages; and, by Murphy's law, `textcomp` did not contain a necessary `\ProvidesPackage` command, with the result that it claimed to be written for a future release[2] — something that would have been caught by any test file exercising the package.

Another embarrassing example of a missing test file in that release was the `\t` error. To better support language files from the Babel suite, some of which make the " character active, we changed all internal definitions of characters and accents from hexadecimal notation, such as `"7F`, to decimal, i.e., to 127 in that case. Unfortunately in the definition for `\t` we did this wrong and `"7F` became 79, giving very strange effects when the accent was used.[3] An error like this would have been automatically caught if we had, for each output encoding, a test file to check that each definition in the encoding results in the 'right' glyph or glyphs.

[2] The technical reason for this behavior, for those who wonder, was that the release date of the package, which is an optional argument to the (missing) `\ProvidesPackage` command, was there but was mistakenly picked up the `\NeedsTeXFormat` which then produced a warning as the release date of `textcomp` was later than the nominal release date (of 1997/12/01) for the format of the distribution.

[3] Both errors got found and reported several times within two days after the release, so the patch release came out quite quickly this time.

3 Call for volunteers

Thus to make the LaTeX system even more reliable we call on you for help! What we hope to find is a new group of volunteers that is interested in working on an extension of the LaTeX test suite system. There is no need to be an expert TeX or LaTeX programmer for this task though some experience with LaTeX and its inner workings will be necessary.

If you are interested in joining this effort, please contact Daniel Flipo at

`Daniel.Flipo@univ-lille1.fr`

who kindly agreed to act as a coordinator between the individual volunteers.

There are a number of areas in which further test files would improve the system enormously. They are outlined in the following sections.

3.1 Testing existing interfaces

Testing existing interfaces is a very important task, one not so far, for several reasons, adequately covered by the test suite. This will not only help us to detect problems when fixing errors in LaTeX but, more importantly, it will help one day in the transition to a new system since these test files will then clearly identify which interfaces are compromised (deliberately or by mistake) by the new system. This in turn will then help to produce, if necessary, procedures to automatically translate source documents from LaTeX to its successor.

What we are looking for are test files that describe and test the current interfaces on all levels. This is certainly an ambitious task, but perhaps also one of the most interesting and rewarding ones within this list.

3.2 Testing corrected bugs

As described above, several of the bugs reported to us have been fixed and a test file showing the correct behavior has been added. But for many this is not the case.

What we are looking for is the provision of test files for all bugs reported and fixed, so that future releases will not by mistake revert any of these fixes without alerting the maintainers. This means working through our bug database and devising test files showing the correct behavior. As we ask submitters of bug reports to send in a test file that shows the incorrect behavior, and they usually do so, it is often possible to start from the submitted file and modify it slightly so that it fits into the regression suite concepts.

3.3 Testing new extensions

What is important for the kernel interfaces is also important for the core packages and extensions: these interfaces should be exercised in such a way that any future changes will be automatically detected. Again this provides interesting mental exercise since it isn't always easy to decide what is pertinent for the interface and how to exercise it so that enough (but not too much) information ends up in the `.log` file.

3.4 Testing contributed packages

A final area which is important is the testing of packages which lie outside the control of the LaTeX maintainers. Although we cannot in all cases guarantee that corrections to the kernel software will not harm any such package, we are, of course, very much concerned to avoid making any change that makes third party packages invalid. In the past, whenever we noticed (or even suspected) such a problem we tried either to avoid it, by choosing a different solution, or, if that was not possible for some reason, to find the maintainers of the package and give them notice of a possible clash so that such problems could be avoided.

There is a problem with testing the interfaces of third party packages: changes by the package author, to either the interface or the implementation of the package, can upset the test suite as easily as can changes to the LaTeX kernel by the LaTeX3 project team. Thus, to avoid our limited time resources being used up in chasing after errors introduced in this way (being neither our fault nor being correctable by us), it would be necessary to develop clear protocols for how this part of the test suite should be maintained, e.g., what requirements a package must fulfill to be included into it, what obligations an author of such a package agrees to, etc. This is not yet done and so it is part of the volunteer effort.

We close our plea for help with a quote taken from [2] which shows how the Grand Wizard sees the task of writing such test files (which does not mean you have to follow his advice):

> To write such a fiendish test routine, one simply gets into a nasty frame of mind and tries to do everything in the unexpected way. Parameters that are normally positive are set negative or zero; borderline cases are pushed to the limit; deliberate errors are made in hopes that the compiler will not be able to recover properly from them.
>
> *Donald Knuth 1984*

References

[1] David Carlisle and Frank Mittelbach. *The LaTeX regression test suite: concepts and implementation. TUGboat*; to appear.

[2] Donald E. Knuth. A torture test for TeX. Report STAN-CS-84-1027, Stanford University, Department of Computer Science, Stanford, CA, USA, 1984.

[3] Leslie Lamport. *LaTeX: A Document Preparation System.* Addison-Wesley, Reading, Massachusetts, second edition, 1994.

◇ Frank Mittelbach
LaTeX3 project
`Frank.Mittelbach@eds.com`

LaTeX News

Issue 8, December 1997

New supported font encodings

Two new font encodings are supported as options to the fontenc package:

OT4 This is a seven-bit encoding designed for Polish. The LaTeX support was developed by Mariusz Olko.

TS1 This is the 'Text Companion Encoding'; it contains symbols designed to be used in text, as opposed to mathematical formulas, and some accents designed for uppercase letters. It is currently supported by the 'tc' fonts, which match the T1 encoded 'ec' text fonts. A subset of the glyphs in this encoding is supported by virtual fonts distributed with the PostScript font metrics on the CTAN archives. (This is the '8c' encoding in Karl Berry's fontname scheme.) The textcomp package provides access to this encoding but here is a warning to current users of that package: some of the internal names for the characters have changed.

New input encodings

These additions to the inputenc package are `decmulti` (the DEC Multinational Character Set, contributed by M. Y. Chartoire) and `cp1250` (an MS-Windows encoding for Central and Eastern Europe, contributed by Marcin Woliński). There is also a `cp1252` encoding that is identical to `ansinew`.

Tools

The calc package (used in many examples in *The LaTeX Companion*) has been contributed to this distribution by Kresten Krab Thorup and Frank Jensen. This is essentially the same as the version that has been available from the CTAN archives for some time, with one minor change: to use LaTeX-style error messages. It enables the use of arithmetic expressions within arguments to standard LaTeX commands where a length or a counter value is required. For example:

```
\setcounter {page} { \value{page} * 2 + 1 }
\parbox { 3in - ( 2mm + \textwidth / 9 ) }
```

There have also been some improvements to several other packages in this collection. In particular, bm now works correctly with constructions such as `\bm{f'}` involving `'` or other characters which use TeX's special "`\mathcode"8000`" feature. Also, multicol sets the length `\columnwidth` to an appropriate value; this enables it to work with classes that support two-column setting, e.g., the AMS classes.

Graphics

The special `oztex.def` driver file has been removed, and OzTeX support has been merged with dvips, following advice from Andrew Trevorrow about OzTeX 3.x.

The keyval package has had some internal improvements: to use LaTeX format error messages; and to avoid '`#` doubling'. This latter change means that the `command` key for the graphicx version of `\includegraphics` should now be used with one `#` rather than two. For example, `command = 'gunzip #1`. Fortunately this key is almost never used in practice, so few if any documents should be affected by this change.

LaTeX3 experimental programming conventions

As announced at the TeX Users Group meeting (Summer 1997), a group of highly experimental packages will soon be released to allow experienced TeX programmers to experiment with, and comment on, a proposed set of syntax conventions and basic data-types that might form the basis for programming large scale projects in TeX. They will be located in this CTAN directory:

```
CTAN:macros/latex/packages/expl3
```

The documentation of this material is as follows: individual package files provide outline, draft documentation; there is an article that gives an overview of the syntax and related concepts; there is a `readme.txt` file containing a brief description of the collection.

All aspects of these packages are liable, indeed likely, to change. They should not be used at this stage for anything that requires a stable system. However, we do encourage people to experiment with these packages, and to send comments on them to the `LaTeX-L` mailing list. To subscribe to this list, mail to:

```
listserv@urz.uni-heidelberg.de
```

the following one line message:

```
subscribe LATEX-L <first-name> <second-name>
```

A Week on Electronic Documents and Typography

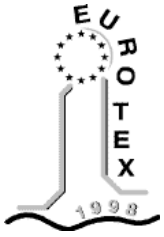
Saint-Malo, France, March 29 – April 3, 1998

The first WEPT *(Week on Electronic Publishing and Typography)* was held at Darmstadt, Germany in April 1994. The second, WEPT'98, will take place in Saint-Malo, France from March 29 to April 3, 1998.

WEPT'98 features, as in 1994, four conferences and sixteen tutorials, open to everybody.

The timetable for the whole week is available at `http://www.irisa.fr/wept98/`

Come and participate and have fun!



EuroTEX'98 – March 29–31
10th European TEX Conference
fax: +33 1 30 87 06 25
email: `eurotex98@ens.fr`
`http://www.ens.fr/gut/manif/eurotex98/`

RIDT'98 — March 30 - April 1st
Raster Imaging and Digital Typography
fax +41 21 693 66 80 — email: `hersch@di.epfl.ch`
`http://diwww.epfl.ch/w3lsp/pub/ridt98/`

PODDP 98

PODDP 98 – March 29-30
Workshop on Principles
Of Digital Document Processing
email: `nicholas@cs.umbc.edu`
`http://www.cs.ust.hk/~dwood/.poddp98/`

EP'98 — April 1-3
7th International Conference
on Electronic Publishing, Document Manipulation
and Typography
fax: +33 2 99 84 73 95 — email: `ep98@irisa.fr`
`http://www.irisa.fr/ep98/`

TUG'98:

"Integrating TEX with the surrounding world"
Uniwersytet Mikołaja Kopernika, Toruń, Poland
17–21 August, 1998

The 1998 TEX Users Group Conference will be organised and hosted by the Polish TEX Users Group GUST. Planning is now well under way, and enquiries may be addressed to `tug-98@mail.tug.org`.

1 Call for Papers

Proposals for papers are now being solicited: preference will be given to papers which deal most directly with the theme of the conference, but papers on related topics (e.g., DSSSL,HTML, PDF, SGML, XML, etc.) are not excluded. Proposals (which should ideally be written in English, but which may be written in another language by prior arrangement) should be sent to the TUG'98 Programme Committee (`tug-98-papers@mail.tug.org`).

Each proposal should include the title, name, address, e-mail address and affiliation (where appropriate) of the proposer, together with an extended abstract (the latter should correspond to approximately one side of A4. In addition, there should be an estimate of the necessary time for verbal presentation (excluding questions: a further five minutes will be allowed for questions after each talk), and a further estimate of the number of printed pages which will be required to reproduce the full text of the article using the `[l]tugproc` macros. Any special needs to process the paper should also be stated (for example, colour pages in the preprints/proceedings; e-TEX, pdfTEX, Omega, etc.).

2 Deadlines

30 January 1998:
Deadline for receipt of proposals

20 February 1998:
Deadline for notification of acceptance/rejection

10 April 1998:
Deadline for receipt of first drafts

29 May 1998:
Deadline for receipt of final versions

For more extensive information, consult the TUG'98 website at `http://www.tug.org/tug-98/`, which also includes a list of related conferences.

3 Getting there

Delegates arriving from overseas will probably fly into Warszawa ("Warsaw") Airport: the schedule for LOT, the national carrier, is on-line at `http://www.lot.com/schedule/`. Many other carriers (e.g., British Airways, KLM, Sabena) also fly into Warszawa.

From the airport, delegates should take the Airport City bus to Warszawa Centralna ("Warsaw Central") station: this bus journey costs 4 zl (the current exchange rate is about 5 zl to 1 pound sterling). The number 175 bus is a cheaper option (cost: 1.6 zl), but not advisable if traveling with large quantities of luggage, exposed wallets, documents, etc. ... Taxis can prove very expensive unless booked by telephone in advance from a reliable company such as Wawa or Wolfra (a useful list of taxi companies with telephone numbers can be found at `http://www.inter.com.pl/warsaw/wttt.htm`.

From the station, trains run direct to Toruń; the PKP (Polish Railways) timetable is on-line at (`http://bahn.hafas.de/bin/db.w97/query.exe/en`).[1] Train travel within Poland is not expensive, and delegates should book first-class accommodation with reserved seats if possible: couchettes and sleeping cars are available on some overnight trains. Beware of pickpockets, and never allow yourself to be forced to pass between two strangers in the train corridor.

If you prefer to drive to Toruń, and are coming from outside Poland, expect considerable delays at such well-known border crossings such as Frankfurt am Oder. Far better is to seek out the little-used local crossings such as that at Kostrzyn. Formalities are minimal, and you need not complete a currency declaration unless you are carrying more than 2000 ECU. Once in Poland, pay particular attention to speed limits, which are strictly enforced (60 kph in towns/villages, 90 kph outside, 110 kph on expressways and motorways: any/all of these may be overridden by signs). The alcohol limit for drivers is zero.

[1] Specify *Warszawa Centralna* as 'From:' and *Torun Glowny* as the destination 'To:'; in fact, any European station can be the starting and/or finishing point.

Calendar

1998

Feb 5 — DANTE TEX–Stammtisch at the Universität Bremen, Germany. For information, contact Martin Schröder (MS@Dream.HB.North.de; telephone +49-421-2239425). *Regular schedule*: First Thursday (if not a holiday), 18:00, Universität Bremen MZH, 28359 Bremen, 4th floor, across from the elevator.

Feb 5 — DANTE TEX–Stammtisch at the Universität Karlsruhe, Germany. For information, contact Klaus Braune (braune@rz.uni-karlsruhe.de; telephone 0721/608-4031). *Regular schedule*: First Thursday (if not a holiday), 19:30, Rechenzentrum der Universität Karlsruhe, Zirkel 2, 3.0G Raum 316.

Feb 6 – 8 — FairPrint '98, 3rd National DTP-Conference on the Application of Computer Science in Printing, Budapest, Hungary. Note: official language Hungarian; if more than 80 foreign participants, simultaneous translation will be provided in English and/or German. For information, contact instantc@mail.datanet.hu.

Feb 12 — DANTE TEX–Stammtisch, Berlin, Germany. For information, contact Rolf Niepraschk (niepraschk@ptb.de). Second Thursday, 19:00, Gaststätte "Bärenschänke" near the U-Bahnhof, "Oranienburger Tor".

Feb 25 – 27 — DANTE TEX-Tagung '98 in Oldenburg, Carl-von-Ossietzky University, Oldenburg, Germany. For information, contact Konrad Blum, kblum@prehp.physik.uni-oldenburg.de.

Feb 24 — DANTE TEX–Stammtisch, Köln, Germany. For information, contact Daniel Schlieper (Daniel.Schlieper@uni-koeln.de) or visit http://www.uni-koeln.de/themen/texmf/lokal/termine.html#stammtisch. Fourth Tuesday, 20:00, ZPR Seminarraum (in basement), Weyertal 80, 50931 Köln.

Mar 5 — DANTE TEX–Stammtisch at the Universität Bremen, Germany. (For details, see Feb 5.)

Mar 5 — DANTE TEX–Stammtisch at the Universität Karlsruhe, (For details, see Feb 5.)

Mar 12 — DANTE TEX–Stammtisch, Berlin, Germany. (For details, see Feb 12.)

Mar 22 – 24 — **TEXNortheast TUG Conference**, TEX/LATEX Now, Loews Hotel, New York City. For information, check the link from the TUG home page, http://www.tug.org, or write to the committee at tex-nyny@ccrwest.org.

Mar 29 – Apr 1 — EuroTEX '98, St. Malo, France. For more information, visit http://www.ens.fr/GUTenberg/manif/\webreak eurotex98/index.html or contact Michele Jouhet (michele.jouhet@cern.ch).

Mar 24 — DANTE TEX–Stammtisch, Köln, Germany. (For details, see Feb 24.)

Apr 1 – 3 — EP 98: Seventh International Conference on Electronic Publishing, Document Manipulation and Typography, St. Malo, France. For information, visit http://www.irisa.fr/ep98/week.html or contact Jacques André (jandre@irisa.fr).

Apr 2 — DANTE TEX–Stammtisch at the Universität Bremen, Germany. (For details, see Feb 5.)

Apr 2 — DANTE TEX–Stammtisch at the Universität Karlsruhe, (For details, see Feb 5.)

Apr 9 — DANTE TEX–Stammtisch, Berlin, Germany. (For details, see Feb 12.)

Apr 20 – 23 — Workshop RLA2C: Speaker Recognition and its Commercial and Forensic Applications, Avignon, France. For information, see http://lia.univ-avignon.fr/RLA2C.

Apr 30 – May 3 — GUST 6th Annual Meeting, Bachotek, Poland: TEX — A Standard for Scientific and Technical Publishing. For information, contact Jola Szelatyńska, mjsz@cc.uni.torun.pl.

Apr 21 — DANTE TEX–Stammtisch, Köln, Germany. (For details, see Feb 24.)

May 7 — DANTE TEX–Stammtisch at the Universität Bremen, Germany. (For details, see Feb 5.)

Status as of 21 December 1997

May 7 — DANTE TEX–Stammtisch at the Universität Karlsruhe, (For details, see Feb 5.)

May 14 — DANTE TEX–Stammtisch, Berlin, Germany. (For details, see Feb 12.)

May 26 — DANTE TEX–Stammtisch, Köln, Germany. (For details, see Feb 24.)

Jun 3 – 5 — SSP 20th Annual Meeting, Society for Scholarly Publishing, San Diego, California. For information, visit `http://www.edoc.com/ssp/` or write to `ssp@resourcenter.com`.

Jun 4 — DANTE TEX–Stammtisch at the Universität Bremen, Germany. (For details, see Feb 5.)

Jun 4 — DANTE TEX–Stammtisch at the Universität Karlsruhe, (For details, see Feb 5.)

Jun 11 — DANTE TEX–Stammtisch, Berlin, Germany. (For details, see Feb 12.)

Jun 23 — DANTE TEX–Stammtisch, Köln, Germany. (For details, see Feb 24.)

Aug 17 – 21 — **TUG'98** — The 19th annual meeting of the TEX Users Group, Torun, Poland: "Integrating TEX with the surrounding world". For information see the call for papers, p. 314, or `http://www.tug.org/tug-98/`.

Oct 30 – Nov 1 — TypeCon '98, Society of Typographic Aficionados, Westborough, Massachusetts. Principal speaker: Matthew Carter. For information, contact Bob Colby, (`bobcolby@tiac.net`).

For additional information on TUG-sponsored events listed above, contact the TUG office (503-223-9994, fax: 503-223-3960, e-mail: `tug@mail.tug.org`). For events sponsored by other organizations, please use the contact address provided.

Late-Breaking News

Production Notes

Mimi Burbank

I feel a little bit like the White Rabbit in "Alice..." — "I'm late! I'm late!..."

Once again, production of this issue entailed considerable "*beta* testing" of software on quite a few different platforms located here at the Supercomputer Computations Research Institute (SCRI) at Florida State University — in preparation for the next release of the **TEX Live** CD-ROM. I might add that working with the new software ranged anywhere from very satisfying to completely exasperating, with not a few hilarious exchanges between myself and Sebastian Rahtz, who is working on the **TEX Live** setup here at SCRI. While he was compiling on the various platforms, the systems people were busy upgrading the systems, and so we had a wonderful mixture of compiled programs, only to begin again the next day. Oh, we also suffered a security breach and the system was closed to all users for 24 hours, and to remote users for three days!

This issue contains two articles which were presented at TUG'97 — the pdfTEX manual by Hàn Thế Thành, and "The LATEX3 Programming Language — a proposed system for TEX macro programming", by the LATEX3 project members.

Output The final camera copy was prepared at SCRI on the following UNIX platforms: IBM rs6000's running AIX v3.2.5, v4.1.4.0, and v4.2; DEC Alphas running alpha-osf3.2 and alpha-osf4.0; SGI workstations running mips irix5.3 and mips irix6.2; and an HP workstation running hppa1.1-hpux10.10, using the *TEX Live* setup (Version 3), which is based on the *Web2c* TEX implementation version 7.2 by Karl Berry and Olaf Weber. PostScript output at 600dpi, using outline fonts, was produced using Radical Eye Software's dvipsk 5.76a, using the `dvips -Pcms` option and printed on a QMS 860 printer.

Coming Next Issue

TUG and the UKTUG are preparing a new **TEX Live** CD-ROM, with an anticipated release date of April, 1998. We currently plan to delay shipping the March issue of *TUGboat* so that this CD can be included in the issue.

We expect to present a couple of METAPOST articles in the next issue — the long awaited manual on "METAFONT: practical and impractical applications" by Boguslaw Jackowski, and an article by Denis Girou entitled "'pst-fill': a PSTricks package for filling and tiling areas". We hope to set the whole of the next issue of *TUGboat* in EM fonts, and to publish technical details about the package.

◇ Mimi Burbank
SCRI, Florida State University,
Tallahassee, FL 32306 – 4130
mimi@scri.fsu.edu

Institutional Members

American Mathematical Society,
Providence, *Rhode Island*

CNRS - IDRIS,
Orsay, *France*

CERN, *Geneva*, *Switzerland*

College of William & Mary,
Department of Computer Science,
Williamsburg, *Virginia*

Communications
Security Establishment,
Department of National Defence,
Ottawa, *Ontario*, *Canada*

CSTUG, *Praha*, *Czech Republic*

Elsevier Science Publishers B.V.,
Amsterdam, *The Netherlands*

Florida State University,
Supercomputer Computations
Research, *Tallahassee*, *Florida*

Hong Kong University of
Science and Technology,
Department of Computer Science,
Hong Kong

Institute for Advanced Study,
Princeton, *New Jersey*

Institute for Defense Analyses,
Center for Communications
Research, *Princeton*, *New Jersey*

Iowa State University,
Computation Center,
Ames, *Iowa*

Los Alamos National Laboratory,
University of California,
Los Alamos, *New Mexico*

Marquette University,
Department of Mathematics,
Statistics and Computer Science,
Milwaukee, *Wisconsin*

Masaryk University,
Faculty of Informatics,
Brno, *Czechoslovakia*

Mathematical Reviews,
American Mathematical Society,
Ann Arbor, *Michigan*

New York University,
Academic Computing Facility,
New York, *New York*

Nippon Telegraph &
Telephone Corporation,
Basic Research Laboratories,
Tokyo, *Japan*

Princeton University,
Department of Mathematics,
Princeton, *New Jersey*

Smithsonian Astrophysical
Observatory, Computation Facility,
Cambridge, *Massachusetts*

Space Telescope Science Institute,
Baltimore, *Maryland*

Springer-Verlag,
Heidelberg, *Germany*

Stanford University,
Computer Science Department,
Stanford, *California*

University of California, Berkeley,
Center for EUV Astrophysics,
Berkeley, *California*

University of California, Irvine,
Information & Computer Science,
Irvine, *California*

University of Canterbury,
Christchurch, *New Zealand*

University College,
Cork, *Ireland*

University of Delaware,
Computing and Network Services,
Newark, *Delaware*

Universität Koblenz–Landau,
Koblenz, *Germany*

University of Oslo,
Institute of Informatics,
Blindern, *Oslo*, *Norway*

University of Stockholm,
Department of Mathematics,
Stockholm, *Sweden*

University of Texas at Austin,
Austin, *Texas*

Università degli Studi di Trieste,
Trieste, *Italy*

Uppsala University,
Uppsala, *Sweden*

Vrije Universiteit,
Amsterdam, *The Netherlands*

Wolters Kluwer,
Dordrecht, *The Netherlands*

Yale University,
Department of Computer Science,
New Haven, *Connecticut*

Leadership in Typesetting

1466 NW Front Avenue,
Suite 3141
Portland, OR 97209

Email: tug@mail.tug.org
Phone: +1 503 223–9994
Fax: +1 503 223–3960

1998 Individual Membership Application

TUG individual membership rates for 1998 are listed below. Please check the appropriate box and mail payment along with a copy of this application to the address below.

	Check one only	Rate	Amount
Annual membership			
includes TUGboat (4 issues)	☐	$60	____
$5 discount if received by 28 Feb 98	☐	$55	____
Student membership (Attach photocopy of 1998 student ID)			
includes TUGboat (4 issues)	☐	$40	____
$5 discount if received by 28 Feb 98	☐	$35	____
Voluntary donations			
General TUG contribution			____
Contribution to Bursary Fund*			____
		Total $	____

Check one: ☐ Payment enclosed ☐ Charge Visa/Mastercard:

Account Number: _____ Exp date: _____

Signature: _____

*The Bursary Fund provides financial assistance to members who wish to attend the TUG Annual Meeting.

Information for TUG membership list

In the space below please provide the information to be posted to our database regarding your membership. Enter all items carefully as this information will be used for mailing publications and other notices to you.

Name: _____

Address: _____

_____

_____

Affiliation: _____

Phone: _____ Fax: _____

Email address: _____

TEX Consulting & Production Services

Information about these services can be obtained from:

TEX Users Group
1466 NW Front Avenue, Suite 3141
Portland, OR 97209-2820, U.S.A.
Phone: +1 503 223-9994
Fax: +1 503 223-3960
Email: tug@mail.tug.org

North America

Anagnostopoulos, Paul C.
Windfall Software,
433 Rutland Street, Carlisle, MA 01741;
(508) 371-2316; greek@windfall.com
We have been typesetting and composing high-quality books and technical Publications since 1989. Most of the books are produced with our own public-domain macro package, ZzTEX, but we consult on all aspects of TEX and book production. We can convert almost any electronic manuscript to TEX. We also develop book and electronic publishing software for DOS and Windows. I am a computer analyst with a Computer Science degree.

Hargreaves, Kathryn
135 Center Hill Road, Plymouth, MA 02360-1364;
(508) 224-2367; letters@cs.umb.edu
I write in TEX, LATEX, METAFONT, MetaPost, PostScript, HTML, C, C++, Java, Awk and Perl. I take special care with mathematics. I also copyedit, proofread, write lively documentation, do spiral binding, program, and hack fonts (METAFONTand Truetype). I design letterforms, ads, newsletters, catalogs, journals/proceedings, and books. I'm a journeyman typographer and began typesetting and designing in 1979. I coauthored *TEX for the Impatient* (Addison-Wesley, 1990) and some psychophysics research papers. I have an MFA in Painting/Sculpture/Graphic Arts and an MSc in Computer Science. On the side, I do some digital type/human vision research.

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.

NAR Associates
817 Holly Drive E. Rt. 10, Annapolis, MD 21401;
(410) 757-5724
Extensive long term experience in TEX book publishing with major publishers, working with authors or publishers to turn electronic copy into attractive books. We offer complete free lance production services, including design, copy editing, art sizing and layout, typesetting and repro production. We specialize in engineering, science, computers, computer graphics, aviation and medicine.

Ogawa, Arthur
40453 Cherokee Oaks Drive,
Three Rivers, CA 93271-9743;
(209) 561-4585
Experienced in book production, macro packages, programming, and consultation. Complete book production from computer-readable copy to camera-ready copy.

Quixote Digital Typography, Don Hosek
555 Guilford, Claremont, CA 91711;
(909) 621-1291; Fax: (909) 625-1342;
dhosek@quixote.com
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.

Southern California PrintCorp
(800) 899-7267 x801
SAVE MONEY on 1-day Linotronic output for journals. Special *TUGboat* offer. Call now for more information.

Southern California PrintCorp
1915 Midwick Drive, Suite B, Altadena, CA 91001
(800) 899-7267 x888, Fax (818) 399-3565,
BBS (818) 398-3567
We have a ten year history providing 24-hour turn-around imagesetting of PostScript files. Call for FREE information on how *TUGboat*-ers can obtain low-cost, fastest available Linotronic publication production services in the U.S.

Type 2000
16 Madrona Avenue, Mill Valley, CA 94941;
(415) 388-8873; Fax: (415) 388-8865
pti@crl.com
$2.50 per page for 2000 DPI TEX and PostScript camera ready output! We provide high quality and fast turnaround to dozens of publishers, journals, authors and consultants who use TEX. Computer Modern, PostScript and METAFONT fonts available. We accept DVI and PostScript files only and output on RC paper. $2.25 per page for 100+ pages, $2.00 per page for 500+ pages; add $.50 per page for PostScript.

Outside North America

TypoTEX Ltd.
Electronical Publishing, Battyány u. 14. Budapest,
Hungary H-1015; (036) 11152 337
Editing and typesetting technical journals and books with TEX from manuscript to camera ready copy. Macro writing, font designing, TEX consulting and teaching.