TUGboat

Volume 24, Number 2 / 2003

TEX Users Group

TUGboat (ISSN 0896-3207) is published by the TEX Users Group.

Memberships and Subscriptions

2004 dues for individual members are as follows:

- Ordinary members: $75.
- Students/Seniors: $45.

The discounted rate of $45 is also available to citizens of countries with modest economies, as detailed on our web site.

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, as well as software distributions and other benefits. Individual membership is open only to named individuals, and carries with it such rights and responsibilities as voting in TUG elections. For membership information, visit the TUG web site: `http://www.tug.org`.

TUGboat subscriptions are available to organizations and others wishing to receive *TUGboat* in a name other than that of an individual. Subscription rates: $85 a year, including air mail delivery.

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 (`office@tug.org`) or see our web site.

TEX is a trademark of the American Mathematical Society.

Printed in U.S.A.

Addresses

General correspondence, payments, etc.
TEX Users Group
P. O. Box 2311
Portland, OR 97208-2311
U.S.A.

Delivery services, parcels, visitors
TEX Users Group
1466 NW Naito Parkway
Suite 3141
Portland, OR 97209-2820
U.S.A.

Telephone

+1 503 223-9994

Fax

+1 503 223-3960

Electronic Mail
(Internet)

General correspondence, membership, subscriptions:
`office@tug.org`

Submissions to *TUGboat*, letters to the Editor:
`TUGboat@tug.org`

Technical support for TEX users:
`support@tug.org`

Contact the Board of Directors:
`board@tug.org`

World Wide Web

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

Problems not resolved?

The TUG Board wants to hear from you: Please email `board@tug.org`.

[printing date: November 2004]

TUGboat

This issue (Vol. 24, No. 2) is the only regular issue of the 2003 volume year. Vol. 24, No. 1 was the TUG 2003 conference proceedings, and No. 3 will be the EuroTEX 2003 proceedings.

We are unfortunately not able to set a definitive schedule for the appearance of the next few issues.

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

Owing to the lateness of the present issue, and the scarcity of material submitted for future issues, suggestions and proposals will be gratefully accepted and processed as received.

Manuscripts should be submitted to a member of the *TUGboat* Editorial Board. Articles of general interest, those not covered by any of the editorial departments listed, and all items submitted on magnetic media or as camera-ready copy should be addressed to the Editor-in-Chief, Barbara Beeton, to the Managing Editor, Robin Laakso, or to the Production Manager, Mimi Burbank.

The *TUGboat* "style files", for use with either `plain` TEX or LATEX, are available from CTAN and the *TUGboat* web site, `http://tug.org/TUGboat`. For authors who have no network access (browser or FTP), they will be sent on request. Send e-mail to `TUGboat@tug.org`, or write or call the TUG office.

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@tug.org`.

TUGboat Editorial Board

Production Team

Other TUG Publications

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 at `tug-pub@tug.org` or in care of the TUG office.

TUGboat Advertising

For information about advertising rates and options, write or call the TUG office, or see our web page: `http://tug.org/TUGboat/advertising.html`.

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.

METAFONT is a trademark of Addison-Wesley Inc.
PostScript is a trademark of Adobe Systems, Inc.
TEX and $\mathcal{A}\mathcal{M}\mathcal{S}$-TEX are trademarks of the American Mathematical Society.
UNIX is a registered trademark of X/Open Co. Ltd.

General Delivery

From the President

Karl Berry

I'm honored to serve as the new TUG president (as of the TEX Users Group 2003 annual meeting), following in the footsteps of many past and present TEX stalwarts (Pierre MacKay, Nelson Beebe, and Christina Thiele come immediately to mind; see `http://tug.org/tugpres.html` for a rogues' gallery).

2003 TUG organizational matters

- TUG once again has an executive director! The board has given Robin Laakso this title, which more accurately reflects her many various tasks and responsibilities. Congratulations, Robin.
- As has been mentioned in electronic postings, we've opened a small online TUG store: `http://tug.org/store/`. Software releases, *TUGboat* issues, and various past memorabilia are available; members get a 10% discount.
- As of 2003, we are officially shipping three issues of *TUGboat* per year. This is to help us get back on track, without publishing ersatz "double" issues. The regular U.S. postal permit for periodicals requires at least quarterly publication; fortunately, our new nonprofit status (see `http://tug.org/tax-exempt/`) gives us lower mailing rates in general, so our mailing costs are not increasing unduly.
- We now ask that members explicitly request the CTAN archive on CD, instead of mailing it to everyone. (This can be requested when joining or renewing, or by mailing `office@tug.org` at any time.) In our surveys, we found a significant proportion of members did not use the CTAN CD's, and so this allows us to save considerably on production and mailing. The CTAN archive is also included on the TEX Collection DVD, which is sent to all members. (We very much hope that the 2004 TEX Collection will be available soon, but can't give a precise date.)

Icelandic TEX user group

A new TEX user group was formed in the summer of 2003 for Icelandic TEX. We congratulate them and wish them much success. The group's web page: `http://www.rhi.hi.is/istex/`

A list of all user groups and information thereof is available at `http://tug.org/usergroups.html`.

Board resignation and appointments

In January 2004, Stephanie Hogue resigned from the board, owing simply to lack of time. We thank her enormously for her efforts on behalf of TEX and TUG over the years.

In my capacity as president, I have appointed several hard-working TEX folks to the board, to fill some of the vacant positions: Jim Hefferon (long-time maintainer of TUG's CTAN node), Gerree Pecht (long-time TEX enthusiast at Princeton), Steve Peter (linguist, translator, and very active TEXxie), and Steve Grathwohl (in charge of TEXnical operations at Duke University Press).

I'd like to thank them all for agreeing to serve. More information about them and the other board members can be found via the web page `http://tug.org/board.html`.

To continue in office, appointed directors must stand in the next regular election, which will be in 2005; you'll find an announcement about that elsewhere in these pages.

Notable TEXnical events

- The MacTEX TUG Technical Working Group was formed at the TUG 2003 meeting, led by Wendy McKay and others. Its goal is to work on action items that would enhance current and future development of TEX on Mac OS X. They are in continuing contact with Apple. There are many interesting items on their web page: `http://tug.org/twg/mactex/`
- John Hobby, author of MetaPost, has delegated bug fixing and general maintenance to a group led by Taco Hoekwater and Hans Hagen. Activities are coordinated via a MetaPost project at `http://metapost.sarovar.org/`. (Sarovar, supported by the TUGIndia user group, is a public software repository site reasonably analogous to `sourceforge.net` and `savannah.gnu.org`.) John retains overall guidance and direction for MetaPost.
- A small update of the TEX Directory Structure document (version 1.1) is available via CTAN and `http://tug.org/tds/`. The most notable changes are better definitions for font map and encoding files.

I will close with a reminder that the TUG board welcomes input or questions at any time; email us at `board@tug.org`.

◇ Karl Berry
`president@tug.org`

Editorial Comments

Barbara Beeton

Adieu, Daniel Taupin

Daniel Taupin, perhaps best known in the TEX community as the creator of MusixTEX, was killed in a mountaineering accident in the Pyrenees on August 26, 2003. His love of the mountains was great, and he has been honored by his colleagues with a commemorative plaque on the Viaduc des Fauvettes (an old railway bridge that provides access to a number of climbing venues), which was restored in large measure through his tenacious urging. A web site in his memory is at `http://www.pyrenees-pireneus.com/taupin.htm`.

I met Daniel at several meetings of TUG and other groups, and enjoyed his company. He held strong opinions, but always had something interesting to say.

Several papers by Daniel were published in *TUGboat*, beginning with the proceedings of the 1993 annual meeting (**14**:3), where his two main hobbies were introduced in two papers, the first on "Using TEX and METAFONT to build complicated maps" — a description of a "tentatively exhaustive catalog of all the 1500 known climbable crags of France outside the high mountains", and the second, "MusicTEX: Using TEX to write polyphonic or instrumental music".

Another memorial to Daniel appears at `http://icking-music-archive.org/Memorial/Taupin/Statements.html`.

Don Knuth to Michael Downes

I received a note from Don Knuth which had been intended for Michael Downes, written just an hour before he read Michael's obituary. Since the note is on a matter of historical record, it's presented here with Don's permission.

10 Sep 03

Dear Michael

Your fine article in AMS Notices (Dec 2002) sent me to my diary re footnote 2 on page 1389.

TEX 2.0 was installed on Saturday 04 January 1986, on Stanford computers via internet connection from my apartment in Boston where Jill & I were on sabbatical . . . the same day as METAFONT 1.0. The diary also says "typed the index of Volume C thru 'Davis'; never took time to get dressed today

– don knuth

The article in question is entitled "TEX and LATEX 2_ε". It briefly describes the history of TEX, its use at the AMS, development of the AMS packages, and some future directions. The cited footnote refers to November 27, 1985, as the date on which TEX 2.0 was released:

> Conjectural; the historical record for this release seems to be unclear. The 11/27/85 date is the date of the last change recorded in `tex82.bug` after the release of version 1.5 and prior to other changes designated as belonging to version 2.1. An announcement by David Fuchs in the March 1986 issue of *TUGboat* stated that "TEX 1.5, when used with the new CM fonts, is officially called TEX 2.0." Should this be interpreted, perhaps, to mean that the release date of 2.0 is the same as for 1.5?

Another honorary degree for DEK

On June 5, 2003, Don Knuth received the Doctor of Science degree from Harvard University. The citation, he observed, was rather poetic:

> Donald Ervin Knuth, Doctor of Science
> Font of digital ingenuity, icon of algorithmic invention, whose artful efforts have programmed the course of a powerful modern science.

Background information in support of the citation can be found in the June 5 edition of the *Harvard Gazette*, at `http://www.news.harvard.edu/gazette/2003/06.05/01-honorary.html`.

Help save the French Imprimerie nationale

The French Imprimerie nationale, an institution broadly equivalent to the US Government Printing Office or the HMSO in the UK, is heir to the centuries-old tradition of French government printing, starting with the Imprimerie royale, set up by Cardinal Richelieu under King Louis XIII in the 17th century, with forerunners from the Renaissance.

This institution is being disbanded by the French government, with no thought to the preservation of its historic heritage — part of the institution is classified as a "historic monument" — other than to pack it into crates for permanent storage. This move is scheduled for the first half of 2005, destination unknown.

A petition is posted at

`http://www.garamonpatrimoine.org`

for the purpose of encouraging the French government to preserve this unmatched resource. Please read and sign it. The following text is excerpted from the petition.

The historic collection of the Imprimerie is a unique, priceless testimony of the history of the written form, from the 16th century to the present. It includes the Cabinet des poinçons, or Punch Room, holding hundreds of thousands of letterform and character punches, for both western and oriental scripts; functional workshops — a foundry, presses for typography, lithography and copper-plate engraving work, stitching and binding — as well as a library with over 30,000 volumes, and the archives of the State printing works. Set up in 1539 by King Francis I, at the same time as the Collège de France, the national center of academic excellence, this collection stands as the memory of specialized know-how and expertise, and as a center for creation, now fated to disappear if its continued survival is not ensured.

This whole must not be scattered or split up, as regards either its contents, or its functions: museum and conservation, typeface creation, publishing and research. It must be released from the oversight of a ministerial department driven by concerns of economic profitability. This heritage must be housed in Paris, held by an institution guaranteed adequate resources, having the capacity to further enlarge and expand it. Better still, it could be set up as a foundation — a controlled, non-profit organization — which would be a dedicated space for conservation, but equally of interfacing with outside elements, and for research. Concurrently, and as of now, measures should be taken to ensure that the transfer of equipment and expertise proceed speedily, using a transition formula, with no interruption to production, conservation, research or training activities.

Priceless artifacts must be saved, but equally persons, skills, a store of knowledge must be safeguarded, that are at risk of being lost to all humankind.

– – * – –

A message from Jef Tombeur to the TYPO-L discussion list (July 28, 2004) communicated this additional information from the French organization Convention typographique and the European Monotype University:

> Just imagine if the Eiffel tower was made by hand. Not just as it is, but even more so — each bolt, each and every component forged and adorned with care, craft and art. So what? Would just be another French monument. The Cabinet des poinçons (punches), and the French Imprimerie nationale library, are something really different. Not just another French landmark. It is your history, since the [16th] century, that is to disappear, buried, forgotten, maybe dispersed. Because it [is] not only French writing, but Chinese, Arabic, Greek, Hebrew, etc., that are concerned: there are punches and fonts for nearly every written language known till the last century. And books in all writings and languages. But there is another way. And that [is] why, we, the [not] for profit organization Convention typographique, and so many [other] organizations, and individuals, from so various countries, ask you to add your name to the petition (on-line : `http://www.garamonpatrimoine.org`, or print it, or ask for a PDF in your own language or a language [you] understand, to be sent to you; Xerox it, and spread the word around).

Avoid obsolescence: Guidelines for LaTeX users

A couple of years ago, a document by Mark Trettin arrived at CTAN, listing a number of things of which a user of LaTeX 2ε should be aware. This document, originally in German, has now been translated into English ("An essential Guide to the dos and don'ts of LaTeX 2ε, or obsolete Commands and Packages, and some more Mistakes to avoid") and several other languages. A search of CTAN for the keyword "l2tabu" will find it. The suggestions here are helpful for even experienced LaTeX users, giving reasons as well as directives.

Hidden TeX use in Germany

Need a personalized train schedule? If you're traveling on the German railway (Deutsche Bahn AG), you can get one — prepared under the covers using TeX. The company providing this service, and a lot of others, has an interesting web site: `http://www.hacon.de/hafas_e/print2web.shtml`. The site `http://www.travelinfosystems.com/palm/query/query-p2w.cgi/en` provides a similar service for the UK.

Bank statements and similar documents are also created in the same way, but only in Germany, as far as I am aware. (Certainly the statements from my bank aren't produced this way.) Thanks to Volker Schaa for calling these services to my attention.

◇ Barbara Beeton
American Mathematical Society
201 Charles Streed
Providence, RI 02904 USA
bnb@ams.org

Daniel Taupin, 1936–2003

It was with great sadness that I heard of Daniel Taupin's fatal fall from the Rochail, in the Oisans region of the French Alps last year, after a successful solo ascent.

I first met Daniel in 1994 at CERN in Geneva, at the inaugural presentation of Omega by Yannis Haralambous and myself. Lots of grey hair going off in all directions, a big booming heavily French-accented voice, strongly held opinions — just who was this guy?

Over time, at EuroTEX and GUTenberg meetings, I got to know Daniel better. He always focused on the details and the exceptional cases. I share this concern and always try to instill it in my software engineering students.

But he didn't just talk. He built working software. His MusixTEX, used to produce many of today's online musical archives, is a highly complex piece of software, that deals with *all* of the exceptions. It can be used to typeset just about any piece of music in the Western musical tradition, and probably more.

Daniel was also well-known in French climbing circles, and played a key rôle in developing standards for sharing popular climbs between the competitive and recreational styles. Courageous and generous to the core, he was even labelled a "dangerous terrorist" by the French state in 1992 for having, with some friends, removed the fixed pitons from the *via ferata* climb of the famed Aiguille du Midi, done to restore the integrity of the climbing experience.

He brought his typesetting and climbing interests together by using TEX and MetaPost together to produce typeset maps of France showing some of the most popular climbs, thereby showing the way forward.

Daniel's death was untimely. My research team here in Sydney had started to examine questions of high-quality typeset music and maps. Just as we were preparing to get in touch with Daniel, with future collaboration in mind, we received a copy of an email from Fabrice Popineau, announcing Daniel's fall.

I last saw Daniel at the Toulouse GUTenberg meeting in 2000. The evening of the banquet, held in Cintegabelle, the village where Lionel Jospin, the French Prime Minister, was also counsellor-general, we visited the church, which holds one of the finest organs in the French south-west. After the talk and recital, Daniel was explaining to us the difficulties of playing an organ that he had recently found in a church near Paris that was not tuned as a well-tempered clavier. Only certain pieces, in certain keys, could be played with success.

As always, this colourful *personnage* had something to say.

Farewell Daniel, you will be missed.

◇ John Plaice

What is TEX?

Douglas Waud

This journal, *TUGboat*, first appeared in 1980 and has been going strong since. It provides a continuous stream of articles which inform the reader on the latest developments in the field. And there is the problem — the journal has been in production so long that readers are often assumed to know the context. This automatically provides a barrier to an outsider who might stumble onto the journal and ask "What's going on here?"

The editors have therefore decided periodically to insert a primer, a brief summary, to let such a newcomer know what we are so excited about.

1 So what is the object of our affections?

The simple answer is 'TEX'. This is formally called a typesetting program, computer software for producing nicely printed output. TEX came about when Professor Donald E. Knuth, at Stanford University, was planning to publish a series of books on computing. He discovered that the classical process of typesetting books was no longer viable and its various replacements were rapidly declining in quality — and so he decided to take a year off to rectify that omission. He got hooked and the year turned into ten. The centerpiece of his final solution was the program TEX. In a nutshell, TEX allows ordinary people to produce beautiful output of top quality.

Now, one unfamiliar with TEX may think "But I can do that with my word processor!" However, that is a misconception. First, the output will generally be a notch below that of TEX. The situation is analogous to good cooking; you may not know what you are missing until you get a taste of a top quality product. The word processors are getting better these days (for example, they are starting to incorporate many of TEX's tricks) but there is still catching up to do.

There is still another significant distinction. A word processor is described as "WYSIWYG". This is an acronym for "What you see is what you get", with the implication that, as you type, the end result will appear precisely on the screen in front of you. The TEX community interprets WYSIWYG tongue-in-cheek as "What you see is *all* you get!" The point is that, once you get spoiled by TEX, you will not be satisfied with anything less than perfection.

The WYSIWYG issue underlies an even more profound theme in TEX. There is a clear division of labor. The job of deciding what you want to say is separated as much as possible from that of deciding what final form it should take. This has several advantages. For example, as I am typing this, I pay no attention to how it will finally come out on the page. That will be determined by TEX and ancillary programs written by people who are far more competent than I to produce an esthetically optimal final result. Now, if I have strong views, special training, or an area off the beaten track where what I want is just not available (rare these days) I can tweak to my heart's content. But I don't have to. If I simply want to "get the job done", I can do just that without distractions as to how everything will or should be printed. The only formatting I do is tell the program things like "This is what the title will be". In the present case, all I had to do was type '\title{What is \TeX?}' which, incidentally, shows you how easy "talking TEX" can be; even if you know nothing whatsoever about TEX, I think you can read that segment of code. You have one foot stuck in the flypaper already!

Now we come to the jewel in the TEX crown — its ability to typeset mathematics properly. In this area there is no competition. The American Mathematical Society, in fact, was one of TEX's principal sponsors; the vast majority of mathematicians, physicists, and other scientists write their papers using TEX.

When TEX was first developed computers were rather primitive compared to today's. Knuth recognized this and built in modularity. Thus, when TEX processes a file, it does not know what sort of printer or display will be used to view the result. Knuth therefore had TEX put its output in a general form which could then be used as input by "drivers" designed to talk to whatever new printer or monitor came along. This intermediate form is called a "device independent file" (with the file extension .dvi). In turn, there are file viewers, like xdvi in GNU/Linux, for displaying such files on a computer screen.

2 Now meet the family

So far, I have been using the term TEX in its original context — the program Knuth wrote to carry out the magic. However, nowadays, the term "TEX" can also imply a whole family of related programs. TEX is just the grand-daddy of a family of related tools. We can list the main cast now.

TEX First, TEX itself has gone through several updates. TEX78 came first, TEX82 next, and the current and final version is TEX90. (You don't

really have to know these details since you will only use the final version now.)

Metafont Next, a critical part of typesetting is the fonts used. Therefore Knuth also pursued the design of typefaces for TEX. The result was a companion program METAFONT which can be used to produce tailored characters. (Again, this is more for information; you will not be jumping into font design your first day!)

LATEX Now we come to LATEX. This is a component designed to shield the author from the details of TEX. Leslie Lamport, the author of LATEX, recognized that, although TEX is very accessible for the sort of person who loves to wallow in things computer, it can be very formidable to, shall we say, more normal folks. Thus he put together a simpler system designed to protect the author still further from the details of the underlying machinery. For example, he allows the author simply to put something that is to to be written exactly as it is typed between paired begin and end verbatim commands and allows the author to completely avoid having to make any decision as to what type face to use, what size, what style, and the like.

History: The original version, LATEX 2.09, has been replaced by LATEX2e.

BibTEX BIBTEX is an ancillary program to help organize bibliographic references.

MakeIndex MakeIndex is another ancillary program, this time for facilitating of creation of an index.

dvips Earlier we mentioned .dvi files as a generic form of output. With time, the need for a general file format was recognized more widely and the company Adobe Systems Inc. created the PostScript language. The TEX community responded with programs like dvips to convert .dvi files to .ps files.

pdfTEX and pdfLATEX The next step is where we are currently. PostScript has been augmented with the "Portable Document Format" (PDF), also a creation of the folks at Adobe. If you have used the Acrobat reader to read a file, you have been looking at a .pdf file. The TEX community has kept up with this develoment with the program pdfTEX and its LATEX variant pdfLATEX. These allow one to use tricks available in the .pdf format, in particular, to create "hyperreferences", links to stuff on the web.

3 What next?

At this point I encourage you to get your feet wet. Specifically, try to create some documents in LATEX (easier than pure TEX but not out on the "bleeding edge" enough to be overwhelming). First I would recommend getting a manual. There are many out there but I would suggest you start with Lamport's original guide [2]. Next you need to have a TEX system on your computer. If you use Linux, life is easy; your distribution will probably already have it installed. If you use Windows or a Macintosh I suggest you start with Flynn's excellent introduction [1]. If you do not have access to back copies of *TUGboat* you can get it online from:

`http://www.tug.org/`
`tex-archive/info/beginlatex/html`

where, in particular, the chapter "Installing TEX" will be a succinct guide to getting a system installed.

One final parting word of advice: start with something simple, perhaps just try an example from Lamport's book, and then add frills one at a time, *i.e.* crawl before walk!.

4 Online column

At the same time they are launching this column, TUG is planning an online journal which will include a column "\begin{here}" which will be directed toward the person first trying to come to grips with TEX. This column will attempt to identify and clarify standard stumbling blocks. This online journal will be available at

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

References

[1] Peter Flynn. Formatting information. *TUGboat*, 23(2):115–237, 2002.

[2] Leslie Lamport. *LATEX: A Document Preparation System.* Addison-Wesley, Reading, MA, 2nd edition, 1994.

◇ Douglas Waud
Department of Pharmacology
University of Massachusetts
Medical School (retired)
17 Lantern Lane, Shrewsbury, MA, USA
`douglas.waud@umassmed.edu`
`http://users.umassmed.edu/douglas.waud/`

Typography

Maths = Typography?

Richard Lawrence

Introduction

This paper is written for a conference with the theme 'Hidden typography'. Broadly the author's interest is in mathematical printing and typesetting in particular. So, as is usual for academic conferences, the author's task is to persuade you, the reader, that there is some connection between the conference subject and the author's personal interest. To see if this can be done it is pertinent to ask a few questions:

* What is typography and so what is hidden typography?
* What is mathematics?
* Is there any typography in mathematics and is there any hidden typography in it?

The last of these questions is the one that is central to the subject of the conference; answers to the other two help to explain the author's answer to it. So to start with the conclusion:

* There is typography in mathematics.
* The typography in written mathematics is not hidden, it is overlooked.
* There is a strong case for saying that written mathematics is a very highly developed example of typography: it may even be possible to say 'Maths = Typography'.

What is typography?

> The art or process of setting and arranging types and printing from them. (*Concise Oxford English Dictionary, 10th edition*, 2001)

> Typography may be defined as the craft of rightly disposing printing material in accordance with specific purpose; of so controlling the type as to aid to the maximum the reader's comprehension of the text. (Stanley Morison, *First principles of typography*, 1951, CUP)

Here are two definitions of typography, one short and written for a general audience, the other longer and written for an audience wanting to know more. Both are pertinent to the purposes of this paper: it is the aspect of 'arranging' or 'rightly disposing' material 'to aid to the maximum the reader's comprehension' that will be emphasized. Some will argue that both these definitions are rather utilitarian and omit any feeling for the art and beauty that typography can bring to the printed document. However the question of beauty in mathematical typography is also addressed.

So what is *hidden* typography? In the best sense it is Beatrice Warde's crystal goblet typography: invisible or unobtrusive but making the reader's task easier and more pleasant. It is design that helps the reader to extract meaning from the written word. This is very much the sense relevant to the printing of maths. Enough people have trouble grappling with the abstraction of maths that it would not be a good idea to add typographical flourishes and quirks to its written form. Good 'crystal goblet' typography is what the complexity of maths typesetting really does need.

What is mathematics?

> The branch of science concerned with number, quantity, and space, either as abstract concepts (pure mathematics) or as applied to physics, engineering, and other subjects (applied mathematics). (*Concise Oxford English Dictionary*, 2001)

Mathematics is its own branch of science (like physics or chemistry) and comes in two forms, pure and applied. Both forms are concerned with 'number, quantity, and space', one in the abstract, one in practical terms. The graphic representation of maths then has to be able to encompass both abstract notions and practical applications if it is to be any use to mathematicians and those who use maths (physicists, engineers, etc.). It has to deal with 'number, quantity, and space'. It also has to be able to describe a whole branch of science (part will not do). As we know the result is that the printing of mathematics is challenging and specialist work largely avoided by many.

Another common view of mathematics, particularly popular with those who use maths (engineers, physicists, etc.) is that it is a *language* that is used to describe physical situations and relationships. Mathematical equations are used to describe the motion of a pendulum, the decay of radioactive waste, the flow of traffic on congested roads, and the relationship between infinitely large groups of objects in multidimensional space. Mathematics is the language that scientists (physical scientists at least) use to communicate their ideas and observations. Like mathematics in the dictionary definition

This paper was presented at the 2003 St. Bride Printing Library Conference on "Hidden Typography", and appears here with permission. The texts of all the talks from the conference can be viewed at `http://www.stbride.org/conference2003/`.

above, the physical scientist is interested in 'number, quantity, and space'. The graphic representation of maths has to reflect this.

Having used the standard trick of looking at a dictionary definition of the subject it may be instructive to consider the popular views of its users and practitioners. A physicist uses a variety of strange machines to investigate the rules that govern the physical world and records observations as mathematical relationships. A biologist grows then experiments on living things in order to understand more about them perhaps using statistics to support arguments and observations. An engineer designs and builds machinery to exploit the discoveries of other scientists and uses approximate equations to predict how the machinery will behave. The mathematician sits and thinks and scribbles and rearranges equations on paper or blackboard. The mathematician has no machinery or plants or animals to work with. The mathematician's only prop in this populist view is the piece of paper or blackboard, or more specifically equations written on these. Mathematics is in some sense the written equations on these surfaces. If this is indeed so, then it can be appreciated that the optimal arrangement of the symbols in the equations is of some consequence.

So we have three views of mathematics: it is a branch of science dealing with the description of the abstract and real and quantity, number, and space; it is a language; and it is written equations. Properly, and not just as a result of undue deference to the majesty of the Oxford English Dictionary, it is only the first of these. But mathematics' very singular distinction is that it is dependent on its own language to communicate it. That language is only easily communicated in its written form (equations on paper or blackboard). While the individual components of an equation can be read out loud and their relative positions can be described, it is not too far-fetched to say that maths is an unpronouncable, even a silent, language. So that its written form, equations, has to be able to communicate matters of 'quantity, number, and space'.

Printing's influence on mathematics

Before pursuing the intellectual argument that written maths involves a lot of typography, it is instructive and interesting to look at the influence of printing on the development of maths. It is also instructive to see the consequences of trying to write mathematics without symbols to understand why the language of mathematics is necessary.

One of the very earliest mathematical works is the *Algebra* of Al-Khowarazimi, a ninth-century scholar in Baghdad. Florian Cajori (*A history of mathematical notation*, 1929, Open Court) quotes a translation of an example from this work:

> What must be the amount of a square, which, when twenty-one dirhems are added to it, becomes equal to the equivalent of ten roots of that square? *Solution*: Halve the number of the roots; the moiety is five. Multiply this by itself; the product is twenty-five. Subtract from this the twenty-one which are connected with the square; the remainder is four. Extract its root; it is two. Subtract this from the moiety of the rots, which is five; the remainder is three. This is the root of the square which you required and the square is nine. Or you may add the root to the moiety of the roots; the sum is seven; this is the root of the square which you sought for, and the square itself is forty-nine.

In modern notation the statement of the problem and its solution is:

$$x^2 + 21 = 10x$$

$$\begin{aligned} \textit{Solution:}\quad x &= 10/2 \pm \sqrt{[(10/2)2 - 21]} \\ &= 5 \pm \sqrt{(25 - 21)} \\ &= 5 \pm \sqrt{4} \\ &= 5 \pm 2 \\ &= 7, 3 \end{aligned}$$

Even if the reader can not follow the mathematical notation, it should be apparent that the version written using symbols is potentially much easier to comprehend. It is enormously more compact. This compactness and its consequences for intelligibility were commented on a long time ago. William Oughtred (quoted by Cajori), an English mathematician promoting his own work in 1647 noted:

> ... Which treatise being not written in the usuall synthetical manner, nor with verbous expressions, but in the inventive way of Analitice, and with symboles or notes of things instead of words, seemed unto many very hard; though indeed it was but their owne diffidence, being scared by the newness of the delivery; and not any difficulty in it selfe. For this specious and symbolicall manner, neither racketh the memory with multiplicity of words, nor chargeth the phantasie with comparing and laying things together; but plainly presenteth to the eye the whole course and processe of every operation and argumentation.

The essence here is Oughtred's observation that by writing 'with symboles or notes of things instead of words' the argument is 'plainly presenteth to the eye'. The 'symboles' used by early mathematicians are dictated by what the printer had available. So in one of the earliest printed maths books, Cardan's *Ars magna* (1545, quoted in Cajori) the author

contents himself with using abbreviations set in the text roman type to express unknowns. Vieta in 1591 (quoted by Cajori) uses single text roman capitals for unknowns. It was René Descartes in 1637 who finally established the use of lower case italic letters for unknowns. He also started the useful distinction of using letters near the beginning of the alphabet for unknown constants and letters at the end of the alphabet for unknown variables. Significantly by this date it was reasonable to expect a printer to have matching roman and italic types that could be set together. Mathematical setting is notorious for the diversity of sorts it exploits. In early mathematical works the choice of these sorts is limited to what the printer has. For example in early printed books on 'algebra' much use is made of a capital R with a scratched tail to denote root, a sort ordinarily deployed in liturgical work to denote 'Response' (e.g. Cardan, 1545).

One of the more amusingly documented discoveries made by an author seeking out unexploited corners of the printer's stocks is the eventual use of bold to denote vector quantities. Electromagnetic theory was undergoing rapid development in the late nineteenth century and this called for the development of notation in mathematics capable of distinguishing quantities which possessed both direction and size (vectors) from other non-directional quantities (scalars). The first attempt used greek type, but this failed because of confusion with other greek symbols. Maxwell in 1873 promoted German (Fraktur) type for the job. Oliver Heaviside (a populariser of Maxwell's work) was the one who started using bold (*Electromagnetic theory*, 1893, Ernest Benn Ltd):

> Maxwell employed German or Gothic type. This was an unfortunate choice, being itself sufficient to prejudice readers against vectorial analysis. Perhaps a few readers who were educated at a commercial academy where the writing of German letters was taught might be able to manage the German vector without much difficulty; but for others it is a work of great pains to form German letters legibly. Nor is the reading of the printed letters an easy matter. Some of them are so much alike that a close scrutiny of them with a glass is needed to distinguish them unless one is lynx-eyed. This is a fatal objection. But, irrespective of this, the flourishing ornamental character of the letters is against legibility. In fact, the German type is so thoroughly unpractical that the Germans themselves are giving it up in favour of the plain Roman characters, which he who runs may read. It is a relic of mediaeval monkery, and is quite unsuited to the present day. Besides there can be little doubt that the prevalent shortsightedness of the German nation has (in great measure) arisen from the character of the printed and written letters employed for so many generations, by inheritance and accumulation. It became racial; cultivated in youth, it was intensified in the adult, and again transmitted to posterity. German letters must go.
>
> Rejecting Germans and Greeks, I formerly used ordinary Roman letters to mean the same as Maxwell's corresponding Germans. They are plain enough, of course; but, as before mentioned, are open to objection. Finally, I found salvation in **Clarendons**, and introduced the use of this kind of type so called, I believe for vectors (*Phil. Mag,*. August, 1886), and have found it thoroughly suitable. It is always in stock; it is very neat; it is perfectly legible (sometimes alarmingly so), and is suitable for use in formulae along with other types, Roman or italic, as the case may be, contrasting and also harmonising well with them.
>
> Sometimes block letters have been used; but it is sufficient merely to look at a mixed formula containing them to see that they are not quite suitable.

This rather long quote illustrates several points about mathematicians and mathematics.

* Mathematicians are quite passionate about how their maths is presented
* Mathematicians do have a keen appreciation of the utility of notation
* Mathematicians have a real (if idiosyncratic) idea of the aesthetics of maths printing

It is also clear from this quote what a real influence the printer can have on advancing mathematical notation and maths and science themselves.

The beauty of mathematics

The somewhat eccentric writings of Oliver Heaviside do show that mathematicians have a very keen sense of what works in the notation they use to convey ideas. Beyond this utilitarian view of notation, mathematicians also appreciate wider principles involved in the proper written display of their work. This appreciation is linked to an appreciation of the maths itself. A mathematician will be very pleased if s/he is able to simplify an argument or equation and render the mathematical content more comprehensible. Mathematicians speak of the 'beauty' of a well-presented and succinct proof or newly found link between branches of mathematics. The quest for such 'beauty' keeps mathematicians busy trying to refine existing proofs. The proof of the Four Colour Map problem (any map can be coloured using only four colours without the same colour adjoining itself) reported a few years ago is acclaimed,

but relying as it does on thousands of hours of computer calculations, it is regarded as very inelegant. There are many mathematicians busy trying to simplify it. The writing of equations that are compact and convey meaning easily is central to 'beautiful' mathematics.

The written language of mathematics is easily given the attributes of compactness and beauty by mathematicians. It is not surprising to see a long and close association between mathematicians and their printers, the mathematicians exploiting all the available special sorts and skills that the printer can provide. There is such density of meaning in the choice of letter, its style (roman, bold, italic), its typeface (serif, sans serif, script, outline), its alphabet (latin, greek, hebrew), its size, its position relative to other characters (subscript, superscript), that the printer's job is very challenging. The challenge is to follow the very detailed requirements of the mathematical author without any understanding of the content.

The curious thing from a typographer's point of view is how little a typographer can contribute once the mathematician has made all the choices necessary to convey the mathematical meaning. It is not too controversial to suggest that the mathematician is in fact his/her own typographer, at least in the matter of writing equations.

Maths = typography?

If mathematicians define the typography of the equations they write and 'beautiful' mathematics is well-presented equations whose meaning shines through the density of notation that they bear, then perhaps mathematics is just a highly refined typographical game? Going back to Stanley Morison's definition of typography as 'so controlling the type as to aid to the maximum the reader's comprehension of the text', then that is exactly what a mathematician does. Given that mathematics' only physical reality (its only props) are written equations, then perhaps maths really is typography.

Modern mathematical typography

Mathematical typesetting has always challenged the printer (Smith: *The printer's grammar* 1755):

> Gentlemen [authors] should be very exact in their Copy, and Compositors as careful in folowing it, that no alterations may ensue after it is composed; since changing and altering work of this nature is more troublesome to a Compositor than can be imagined by one that has no tolerable knowledge of Printing. Hence it is, that very few Compositors are fond of Algebra, and rather chuse to be employed on plain work, tho' less profitable to them than the former; because it is disagreeable and injures the habit of an expeditious Compositor.

It is not only challenging, it is also expensive. In the 1970s and 1980s publishers sought cheaper alternatives to hot-metal typesetting and used strike-on systems (IBM, Varityper) extensively. Mathematicians grumbled, but mostly accepted arguments about costs. One however was so appalled by the standards of typesetting from Varitypers that he rebelled, spent time studying typography and typesetting, exploited the just-available raster-scan typesetters, and wrote his own type design and typesetting programs. That is Donald Knuth, Professor of Computer Science at Stanford. His type design program is Metafont and the typesetting program is TEX. This is not the place to go into the workings of TEX, but it is interesting to remark that the program is a very good example of what computer scientists call an 'expert system' that is modelled to some degree on the workings of the Monotype sytem that it replaces. Typographers hated it because of its associated typeface, Computer Modern Roman which Knuth modelled on the maths books of his youth (set in Modern Series 7). Typesetters couldn't understand its input coding: this was not modelled on the mechanical requirements of the Monotype system, but is written to make sense to mathematicians. The program was also free. It has been hugely successful to the point that it is the only word-processing system any self-respecting mathematician will use. Typesetters have then found the economic necessity of dealing with it. Several are able to do something about its associated typeface, which pleases typographers.

One significant aspect of TEX that is illustrative of the theme of this paper is a little-remarked feature of it. Knuth wrote an output program abbreviated to 'dvi' which allows TEX to be printed on anything from a 64-pin dot-matrix printer to a laser raster typesetting machine. In all cases the relative positioning of the characters of the equations are perfectly maintained irrespective of the printer used. A mathematician can email a dvi file to a colleague on the other side of the world know exactly what that colleague will see ('dvi' stands for device independent). The exact arrangement of characters is vital to the meaning of the equations. The exact *typography* is vital. For Knuth *typography is maths.*

◇ Richard Lawrence
18 Bloomfield Avenue
Bath BA2 3AB, UK
ZRLawrence@aol.com

On musical typesetting: Sonata for TEX and METAFONT, Op. 2

Federico Garcia

This article appears in the same TUGboat issue as an obituary of Daniel Taupin, author of MusiXTEX, who unexpectedly passed away in 2003. I'd like to take the opportunity to offer it as an homage to his memory.—FG

Modern software industry has provided tools for the typesetting of music for some time already. Two programs in particular — Finale and the more recent Sibelius — virtually exhaust the music community: almost all musicians use either one or the other. Both are WYSIWYG (what you see is what you get) applications, and both have reached that stage of development in which new versions consist of little more than new shortcuts, 'cookies' of questionable relevance, and so on. Competition for ever wider markets, on the other hand, has made these programs horribly 'intelligent': intended for standard situations and the less-than-careful user, they make customization beyond a certain point painstakingly bothersome.

The problem is that non-standard situations are the order of the day in music. On the one hand, there is the so-called 'new music', the contemporary form of the 'art music' tradition, whose notational requirements have proven to be hard to standardize at all;[1] traditional notational practices are quite sufficient for popular and commercial music (increasingly the target of commercial software), but 'new music' overwhelms them in infinite ways. On the other hand, there is musicology, in which not the composition of new but the analysis of old music might potentially require not-yet-conceived-of typographical upheavals — to circle or otherwise highlight certain notes, to tie them in unusual ways, to superimpose staves, ... Thus, any serious system of professional musical typesetting has to give up the hope of foreseeing all needs and possibilities. In a word, it has to be *programmable*.

This is what some years ago got me started thinking of TEX as a suitable environment for such a system. Recently, after having had some experience with MusiXTEX — the 'officially approved' (i.e., acknowledged by Donald Knuth in his web page) application of TEX to music — as well as a taste of what Lilypond has to offer, I have put hands to work on creating (yet another) system, with portability, flexibility and adaptability as priorities.[2] What follows is a report of what I have done so far. If I'm submitting it to the public, it's because I think I might be onto something.

I wrote this article in two stages. First (Op. 1) when I had just devised the main general model, and anticipated the main algorithms needed to implement it. That was toward the end of 2002. And then now, mid-2004, when publishing it in *TUGboat* became really plausible. Particularly the first part (a history of the main idea of the system) features a narrative approach, so I have avoided anachronism: what I know now but didn't know then should not interfere too much. Along the way of telling this story, there is a detailed review and 'refutation' of MusiXTEX. It's admittedly sharp, but sincerely well-minded and grateful.

Then, in the second part, I allowed myself to 'upgrade' the article a little more deeply, taking advantage of these two years of actual work of programming. It's interesting to see how getting concrete changes ideas in unexpected ways.

I think this article can be read without knowledge of either music, TEX programming, or METAFONT. Experience would surely make it easier to understand some lines here and there, but I've tried to make it all clear to any reader, above all to non-musicians. I myself wasn't very experienced with TEX, METAFONT, or musical typesetting — and the bulk of this article is re-telling the steps I've followed and what I've learned from scratch.

PART I: *Qualitative design*

1 The nature of musical typesetting (I)

It's no coincidence that music typesetting programs are, by far, closer to being graphic utilities than text processors. Musical text is multi-dimensional. As

[1] Perhaps I should remind the reader that today's music, like other arts, is far removed from the conventions and traditions of the 'common-practice period' (17^{th}–19^{th} centuries) — for which, incidentally, the 'standard' notation was developed. Everything has changed, from the nature of the musical sound itself to its graphic representation.

[2] MusiXTEX is Daniel Taupin's development of what was originally a joint effort with Andreas Egler, MusicTEX.

Lilypond is the TEX-related but independent system developed by Jan Nieuwenhuizen and Han-Wen Nienhuys. I am by no means familiar with Lilypond — in part because in 2002, when I wrote a first version of this article, I had to give up trying to install it under Windows. One might think that I should have become familiar with it before embarking on the creation of TEX*muse*; but my intuition is that there is no real overlap between the two. For one thing, Lilypond is not a TEX package; for another, it boasts (rightly) of beautiful fonts. TEX*muse*, on the contrary, boasts of not having fonts at all, and this shows how different their nature and approach are. I have grown convinced that, as explained later on, not having fonts is actually the best way to the flexibility I am aiming at, as a musician and as a programmer.

we all know from school, the horizontal dimension is time, and the vertical dimension is pitch. But that's only part of the matter, since different voices come in different staves (different vertical position). The horizontal position of a note in a given staff depends not only on what notes in the same voice come before or after, but also, and typographically most importantly, on what notes are played at the same time by other voices. There is in effect a third dimension involved. (A look at Figure 1 might be in place to visualize all this. This piece being for piano, each staff corresponds to one hand.)

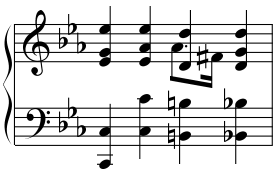


Figure 1: A fragment of a typical score (m. 9 of Chopin's piano prelude in C minor, Op. 28 No. 20).

Music is thus not a line of text (in which only the horizontal dimension matters), but also not a mathematical formula (in which the vertical dimension plays a role): it is an array of several two-dimensional 'strings' that are mutually lined up. Typographically, what this resembles most is, of course, a table. And yet there is an important difference: a musical score usually goes on and on, for many pages — the table has to be broken. Not broken as longtable breaks tables, because in this case the table is not too tall, it is too long. It has hundreds and hundreds of columns, and there is no way to know beforehand how many of them fit in each page.

2 The problem of horizontal spacing and line breaking

An idea that suggests itself naturally when one meditates about this kind of page-breaking is an analogy with TEX's output routine. Instead of collecting lines and deciding when to break the page 'vertically', here the task consists of collecting columns and deciding on horizontal breaks. This would indeed be the case when the table is input in the form of columns. Page-breaking when the table is thus pre-set is actually a rather simple matter, and we can see a solution in action in the workings of MusiXTEX. In fact, that's the latter's main breakthrough: "to address the above aim, a three pass system was developed" [8, p. 9].

The 'three-pass system' works through an external program (MusixFlex), that reads a file (`.mx1`) created by TEX, decides on page- (and, which is analogous, system-) breaking, and writes another file (`.mx2`) for TEX to read during a second TEX pass.

I always wondered whether invoking an external program was really necessary. TEX's handling of horizontal spacing, a substantial part of the line-breaking algorithm for which Knuth takes and deserves much credit — "this, in fact, is probably the most interesting aspect of the whole TEX system" [3, p. 94] — seems so powerful, so flexible, and intuitively so fit to musical spacing, that MusiXTEX appeared to be missing out. I had the impression that music offers an opportunity to really take advantage of this superb algorithm, whose capabilities go far beyond assigning a little extra space to periods and question marks in horizontal mode...

The authors of MusiXTEX lay aside this algorithm on the grounds that it

> implicitly assumes that a normal line of text will contain many words, so that inter-word glue need not stretch or shrink too much to justify the line. This strategy does not work very well for music. If *each bar of music is treated as a word*, in the sense that inter-bar glue is placed at the end of each bar, then the usual result is the appearance of unsightly gaps before each bar rule. This follows naturally from the fact that the number of bars per line is normally many fewer than the number of words in a line of text. [8, p. 9, my emphasis]

This is, in the main, correct. What is flawed is the implicit analogy: it is not "each bar of music" which has to be "treated as a word", because its inner components are also subject to spacing. Since the defining characteristic of a 'word', as far as the line-breaking algorithm is concerned, is that its components remain packed together with rigid space in between, measures (bars) do not qualify as words.

Therefore there is still a possibility of using TEX's line-breaking algorithm. My whole endeavor has been ultimately moved by the intuition that there must be some way to apply it. As we shall see, this *idée fixe* led me — round-about and through a faulty assumption — to the discovery of a promising possibility. But first, now that I have touched MusiXTEX, let me complete my review of it.

3 A word on MusiXTEX

I said above that the problem of horizontally breaking a table is a simple one when the table's columns are input as such: as columns. And that that was the case with MusiXTEX. In fact,

> the fundamental *macro* [of MusiXTEX is] `\notes ... & ... & ... \enotes` where the character

> `&` is used to separate the notes ... to be typeset on the respective staff [*sic*] of the various instruments, starting from the bottom. [7, p. 213]

So you input the first note of the bottom-most staff, then use the magical `&`, then the first note of the second-to-bottom staff, another `&`, the note in the third-to-bottom staff, etc. After getting to the top staff, you start again by going back to the bottom staff, and inputting its second note. The process is exactly analogous to LaTeX's `tabular`, only rotated 90°.

A way to picture what is involved here is imagining that to type a paragraph of text you have to type *a*) all the first words, line by line, *b*) all the second words, *c*) all the third words, *d*) ... It is terribly unnatural, and, as a result, the input is as unreadable as it can get; both creating and reading it is an excellent, but unwelcome, exercise in abstract thought. Because music, after all, is thought of much as text is: as horizontal lines. Many, interconnected lines, but lines still.[3]

Writing a MusiXTeX file is then most uncomfortable, and editing it can be really challenging. But consider the task of 'extracting parts' (selecting, for separate printing, a subset of the staves involved). This is a very common necessity, for players of an ensemble need only *their* staff (not the whole score), and the ability to do it automatically is probably the most important advantage of using a computer for music typesetting. With MusiXTeX's kind of input, this is simply unthinkable (it is analogous to, but more complicated than, extracting a single column from a `tabular` environment, even a simple one; there is no way to copy-paste, not even to automatically find the relevant pieces of the input).

David Salomon says that "the most important feature of TeX is its line-breaking capabilities.... The second most important feature of TeX is its programmability" [6, p. x]. As I have noted, MusiXTeX ignores the first feature; but the way it neglects the second one is actually more exasperating. I started this article by complaining about how customization is disregarded in commercial music software. For example, three steps are needed to change a note-head from its standard shape (the elliptical spot •), say into a ◇ or a × (both fairly usual nowadays). Of course, if you happen to need *many* such changes, you are doomed to follow the same three steps over and over...

What is this like in MusiXTeX? Well, by itself, MusiXTeX cannot do it. You have to go to `musixtex.tex` (naturally a very intricate file), find the relevant definitions, and create your new macros. Foremost, you have to know what you are doing — after all, you are programming TeX at a fairly low level. No hope of doing it if you are no TeXnician.[4] Let alone trying to change temporarily the number of lines in a staff, or creating non-standard key signatures, or connecting two notes from different staves — all of which is excruciating, but at least possible, in Finale and Sibelius. Requiring the user to verticalize what is naturally thought of horizontally is awkward; but prohibiting an efficient and logical programming is — it seems to me — not far short of deserting the very spirit of TeX.

In addition, little care was put in the choice of command names — some bear '`d`' and '`u`' for English *down* and *up*, others '`b`' and '`h`' for French *basse* and *haute*; musical names are applied wrongly ('accent' for articulation, '*sforzando*' for accent, '*pizzicato*' for *staccato*, etc.) — trying to memorize this could mean forgetting what's what. MusiXTeX is not flexible (the limitations of slurring, for example, are severe); it's not even efficient: only nine (or twelve with an 'extension library') staves can be included, because TeX registers are used for almost everything, and they are exhausted quickly. And, this review already becoming far more odious than my intention, I have to say that, with MusiXTeX, quality — O precious treasure and pride of TeX lovers — is poor: the output is ugly.[5]

To sum up, MusiXTeX does not in my view take advantage of TeX's unique possibilities. Its typesetting of non-standard music is as demanding as that of WYSIWYG commercial programs, adding the intricacy of an unreadable input. Sadly, I have had to recommend musicians (even those few well acquainted with TeX) to use Finale or Sibelius rather

[3] Taupin devotes section 1.1.1 of MusiXTeX's manual to argue that "the humanly logical way of coding music" is the procedure described. This claim is as wrong as it is apodeictic. He draws from his perception of what reading keyboard music is like. Even here his point is questionable, because the musician reads horizontal chunks from every staff (the whole tradition of sight-reading training is based on this assumption). But, in any case, the composer and the typist (who really matter) certainly do *not* conceive of music vertically. I have never seen anybody setting a score by columns, but, again, by chunks of horizontal material. Incomplete drafts by composers throughout history, including Bach's *Die Kunst der Fugue* (one of the first MusiXTeX projects), prove this. In my view, Taupin is here proving what he believes, rather than believing what he proves: understandably, his system was "the humanly logical way" of *programming* TeX for the task.

[4] The authors do provide an 'extension library' with ◇, ×, and other symbols implemented as note-heads; but this kind of *ad hoc* procedure is not a solution, for the possibilities are still limited.

[5] And this is why Lilypond puts so much emphasis on the beauty of its fonts (see note 2).

than MusiXTEX: the result is better and the effort is less if the situation is standard, and about the same otherwise.

I'm not being just politically correct, however, when I also have to say that, all in all, it is a wonder that MusiXTEX was programmed at all. I regard its creators with deep respect and admiration.

4 Premises for an alternative

The discussion above sets the grounds for what an ideal system would be like. These are my basic premises:

Programmability and flexibility. This is 'simply' a matter of constantly bewaring of rigid designs in programming.

TEX's glue has to be used. Somehow. It seems so fit for musical spacing...

Horizontal input is necessary if we want a natural and logical system, one that can compete, on ease of use, with Finale and Sibelius.

TEX's sufficiency No PostScript. Come on, no cheating, please.

About the first two points, I think it can be said that MusiXTEX simply slipped down the wrong way — far greater difficulties were being tackled by its authors, so we can forgive them.

But the third premise is a Pandora's box. 'Horizontal input' means that the user will input the different lines (staves), one by one. But there is nothing in a given particular line that indicates the relationship of its notes to the notes in other lines (and, remember, the horizontal position of a note depends foremost on the notes in other staves). For example, there is no way to know, by only reading line 3, whether its fifth note will be played before, at the same time, or after (graphically, to the left, directly above, or to the right of) the tenth note of line 6.

MusiXTEX solves the problem, as mentioned, by asking the user to input *columns*: it is the user who figures the vertical relationships out. But this is precisely what we want to spare the user. Could TEX possibly do it? Well, since these relationships depend on (and only on) the rhythmic characteristics of the notes involved, to figure them out TEX would need to 'understand' those rhythmic characteristics: nothing less than being taught music.

I still tried something else before committing to TEX's musical instruction: if the problem is to horizontalize a vertical input, why not simply rotate the score? The user will type staff by staff, as is natural to him, but for TEX this will mean column by column, which is natural for it. In addition, the problem of page-breaking a horizontally long table becomes the more manageable (and already solved) one of page-breaking a *vertically* long one. Even TEX's glue would be automatically recovered, only it would apply now to staves (rather than measures), which do behave as words in the sense that inner space is not stretched.

I was thrilled by this idea. I still think it is worth exploring for a table-like approach like MusiXTEX. But I soon realized this does not solve the big problem. The input is simplified, but the user still has to figure out the (now horizontal) rhythmic correspondences. And, again, TEX's spacing algorithm would be under-used, being applied to the relatively unimportant problem of inter-staff spacing. The output routine, on the other hand, has nothing resembling space factors with which to work on inter-note spacing (which was my main motivation to think about that algorithm).

So, no way out: for the third premise, TEX had to take music lessons, and I would be the appointed teacher. I was scared, and maybe that, along with my taste for TEX's line-breaking algorithm, is why I focused first on the second premise: bringing back TEX's glue. It will be seen that this actually involved a logical slip — but a fortunate one, because by tackling the wrong problem I found a solution to the real one.

What about the fourth premise? Why did I set this 'only-TEX' requirement to my system? I don't really know — it must have been sheer love of TEX.[6]

5 Bringing glue back

The intuition that TEX's spacing is useful for music can be stated basically thus: different (rhythmic) kinds of musical notes receive different amounts of space to their right, similar to the way periods, commas, and other characters, do in text.[7] A succession of notes can thus be imagined as a succession of words, separated by blank spaces; these spaces will be stretched by TEX (in the same way TEX

[6] And total ignorance of PostScript! (although I am willing to learn if it proves necessary; Van Zandt's miraculous PSTricks is a compelling taste of what can be achieved by 'cheating').

[7] A quick summary of the algorithm: TEX calculates how much the line has to be stretched to reach the right margin (to be justified). Then this total extra space is distributed between all the blank spaces — elastic spaces, TEX's 'glue' — in the line, proportionately to the 'space factor' of the characters before the spaces. Periods, for example, have a larger space factor than letters, and therefore spaces after periods will be given a bigger share of the total stretching.

Throughout this article I consider the algorithm only in its 'stretching' aspect, and without interference from infinite glue or 'badness' parameters.

stretches spaces in a normal line of text), so that if the space factors assigned to each note 'map' their rhythmical nature, the final stretching will agree to the rules of musical spacing.

So, the fruitful analogy is words = notes (as opposed to words = measures, which is 'semiotically' more natural, but typographically misleading — see section 2).[8] Another important aspect of this analogy will be treated in the following section.

However, it will be noted that the intuition paraphrased above is not perfectly rigorous. In all truth, as has been already said, the space at the right of the musical notes depends *not only* on their own rhythmic characteristics, but above all on the notes (which, and how many) present in other staves. The assumption that this spacing behaves similarly to the spacing for justification of texts holds true only in two cases: *a*) when there is only one staff; and *b*) when the note in question is the smallest (rhythmically) of all notes played by the different staves.[9]

OK, I can state *b*) now, with an *a posteriori* conceptualization. At the time I was struggling with 'bringing back TEX's glue', I could not possibly have thought of this as rigorously. Had I considered spacing for many voices, I myself would have probably abandoned the analogy with glue-spacing. But any thinking about many voices was then for me related to 'teaching music to TEX', which was a task I had decided to postpone. Thus I had automatically reduced the scope to one-voice situations — case *a*) — inadvertently making the intuition (and the analogy) true and useful.

An incomplete intuition generated an incomplete consequence: the analogy words = notes (which is right but not sufficient). But only through a further development of this consequence could I start imagining a complete strategy that would solve the *actual* problem of music typesetting, namely the problem of how to deal with polyphony.

6 The nature of musical typesetting (II)

> A page of music type consists of a great many small pieces joined together to represent continuous lines and characters. For instance, this group of notes  is composed of twenty-four pieces, thus:  .

This is how F. H. Gilson [2, p. 11] illustrated, back in 1885, the 'microstructure' of a musical text. In it we can see how much sense it makes to say that a note, in all its atomic character, is analogous to a word (rather than, for example, to a letter). Gilson analyzes his three notes into a series of smaller elements: these elements would be well thought of as letters. Indeed, Finale and Sibelius build up notes from several elements: note-heads, flags (𝅘𝅥) for short values, etc. So does MusiXTEX, whose fonts do not include complete notes at all.[10]

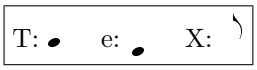


Table 1: Three musical 'letters'.

Notes, then, do seem to behave as words: aggregates of letters. For example, imagine a font including the three characters in Table 1. The word `TeX` would produce 𝅘𝅥.[11]

Now, there are some important things to note:

- Many letters of this musical 'alphabet' would have no width (such as the 'letters' `T` and `e` in the example), since musical 'words' would tend to be vertical rather than horizontal arrangements. In fact, probably the most appealing effect of 'rotating the score' (page 173) is that, when rotated, musical 'words' such as 𝅘𝅥 (now 𝅝) *really* resemble text. But with an unrotated score and zero-width characters, any disturbances of 'normal' vertical alignment (that occur frequently enough in music) would require the manual insertion of rigid space.
- In ordinary text, the space after a word depends on the space factor of *one* of its characters (generally the last one). In music, the space depends on the word — the note — *in its entirety*. The musical letter '𝅗𝅥', for example, will presumably be part of many notes, each of them requiring different spacing. But this is different from, say, what happens with character '.' in ordinary text: the spacing of the whole word is determined by that single character, *on its own and always in the same way*. In our musical font,

[8] I'm using the term 'word' in a TEXnical way: it means any succession of characters ('letters') bounded by spaces (or `\par`'s). So '`No!.`' is a four-letter word ('`!`' and '`.`' being two of the 'letters').

[9] In other words: when the note in question is the smallest in the whole polyphony, it follows that the next note will be present in the same staff. Then, the only consideration for its spacing is its own rhythmical nature — there will be no need to make room for intervening notes in other staves. For a 'special case' of the rule, pointed out by the authors of MusiXTEX, see note 20.

[10] With the annoying result that it is not possible easily to insert a note in ordinary text. Even the authors must have regretted it when they had to do heavy code for this relatively simple need when writing the program's manual.

[11] The vertical line missing here — called 'stem' — would be directly drawn by today's programs; Gilson included it as part of the types.

characters would have to be assigned different space factors each time they are used, which is kind of missing the point.

- The only difference between `T` and `e` (. and .) is the vertical position of the little ellipse. This is of course not the best solution — an infinite alphabet would be needed to put note-heads anywhere in the score. The musical alphabet should have only one letter for the note-head, and move it up and down as necessary.

Taking all this into account, our word would no longer be '`TeX`', but something along the lines of (in LaTeX) '`T\raisebox{-1ex}{T}\sfactor`\X X'. Much less attractive!

7 The advent of METAFONT

There are other reasons why this approach wouldn't work, but at the time I was unable to see them. I think it's the 'aesthetic' disappointment that left me unsatisfied. I rejected the approach, 'arbitrarily' if you will, by the feeling that an originally elegant use of TeX's properties had become a ugly series of *ad hoc* procedures.

I haven't spoken of METAFONT yet. But the truth is that, all along, I had been training in METAFONT — after all, I would soon be forced to create the fonts for my system. So every time I got tired of thinking of TeX, I would go for a METAFONT promenade. I was creating a musical font, imitating Finale's 'maestro'. Many of the symbols used here for musical examples actually come from those early, unexperienced trials. (That's partly why they are not very refined.) The point is that I was increasingly getting to know and love METAFONT, and many times I found myself, after having learned to take good advantage of a new resource, thinking 'Oh, if only TeX could do this...' This was mainly in connection to arithmetic and the use of variables, but in general a feeling was growing that METAFONT was curiously well fitted to the kind of graphic handling required by musical text.

What would have made one think that METAFONT would be fitter for music than graphic utilities? It was indeed curious. But then one day — it was the summer of 2002 — I made a slightly different comparison, and the whole business became defining and foundational. I would actually say that's the moment TeX*muse* was born. The point is not that METAFONT is fitter than PostScript, or than PSTricks, or than PiCTeX. *Truly* relevant is the fact that

METAFONT is fitter *than TeX*

I'm not claiming priority for this. Back in 1992, it occurred to Tom [Thomas E. Leathrum] that it might be possible to take advantage of the fact that METAFONT is *designed* for drawing things. [5, p. 1]

From this occurrence, Leathrum eventually developed the program mfpic.[12] It took me a relatively long time (probably because of my *idée fixe* of applying TeX's glue), but I finally realized the rather obvious fact that building notes up from elements was much more of a 'drawing-things' than of a 'compiling-words' business. Then, Leathrum's solution came to my mind automatically: building the notes is a graphic activity, and it should be done by METAFONT, not by TeX. The latter only gives directions.

8 TeX*muse*

So, that's it: TeX collects the information for the notes (how many note-heads, what kind, where to put them, the direction of their stems, additional signs, etc.), and writes a METAFONT program; running it, METAFONT creates a character for each note. After that, everything is trivial: each note will be like a single-letter word, with its own space factor. TeX applies its stretching algorithm and, almost without knowing it, spaces the music automatically:

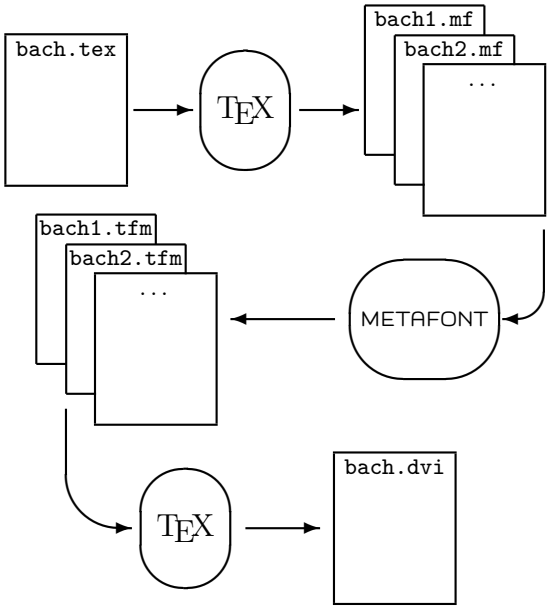


Figure 2: TeX*muse* at work.

[12] Ramón Casares too had a similar idea in 1994. His METATeX is a more direct application of it, different from mfpic (and TeX*muse*) in that it requires the user to know how to use METAFONT — it "only builds the necessary bridges to use TeX and METAFONT in a cooperative way" [1, p. 317].

With this idea, I finally had the outline of a system that seemed to work and fulfill the needs of musical typesetting. This outline subsequently succeeded at all the tests that had beaten all other systems and ideas (the many-voice problem, the four premises, and some other needs of music I haven't mentioned yet). I never performed these tests in any 'rigorous' way, but I qualitatively (and more or less intuitively) felt that the system would hold fair. That's when I went ahead and thought of a name, designed a logo, and wrote an article about it, toward the end of 2002. That article has formed the basis of the present account, and I've so far respected most of its contents and its structure, only struggling to make it more clear.

Let's have a look at how TEX*muse* deals with the mentioned challenges.

8.1 The many-voice problem

It has already been decided that TEX will be taught the secrets of musical rhythm. This means that, reading the horizontal input for each voice, it will know what notes start sounding at any given time — which, typographically, means knowing what notes, and in what staves, are present in any given vertical axis. So, in programming METAFONT, TEX is not limited to building up single notes in each character: instead, it is able to write a program that includes *all* the notes of a vertical axis in a single, rather tall character. Since TEX also knows what the next note is (anywhere in the score), it can figure out the shortest rhythmic value of the current character — which, typographically, determines the spacing and the character's space factor. A correct application of TEX's glue — case *b*) on page 174 — is then possible.

With this the journey is complete: from the wrong analogy 'words=measures', through the incomplete halfway-station 'words=notes', we have arrived at the solution 'words=moments of music'; from the score as a table exasperatingly input by columns, through the score as a table impossibly input by lines, to the score as a single line of text with words automatically designed.

This raises only a minor (but potentially devastating) concern: imagine a score with 24 staves, normal for a regular-size orchestra. A character including notes for 24 staves can be several inches tall, and will usually fill a page up. Is METAFONT really able to handle such tall characters? Luckily, METAFONT allows a maximum height of $2048pt\#$ [4, p. 316]: about 28.5 inches — enough for most needs.[13]

[13] The largest score produced by regular, home-computing, is usually set in pages of the 'Folio' size: 11×17. I have certainly seen larger scores (some, but only a few, taller than 28 inches). But to be noted is that *all* of them are set in handwriting.

The other restriction on character height is that it can be up to 16 times the design size. This could lead to fonts of design size as large as 72pt (for a 16-inch-tall score). I can't foresee any problem with that.

8.2 Premises and promises

The two middle premises (no Aristotelian pun intended) of section 4, namely TEX's glue and horizontal input, were the driving impulses behind the devising of the system in the first place — we can assume they are met. About 'only-TEX', OK, now METAFONT is involved, but we will all agree that there is no cheating there. Look at the root directory of the TDS (TEX Directory Structure): it's called `texmf`. METAFONT is 'part of the family', and in any case it was always intended to be the source of any associated fonts.

Flexibility, on the other hand, is much favored by a system that has no fixed alphabet of graphic elements but constantly creates its own. Thanks to this, the 'core' of TEX*muse* — what it will have built in — can reach a level of flexibility that ensures that only the industrious user will have to resort to lower-level programming. And then, good documentation and an intelligent use of literate programming for the METAFONT part of TEX*muse* (as well as for the TEX part, needless to say) will enable the industrious to further customize the system.[14]

8.3 Other needs

The main idea of TEX*muse* was found by realizing that verticality is, typographically, the main substance of music. But, musically, horizontal constructions are also of the utmost importance. The beams that connect eighth-notes, or the slurs and ties that group notes to indicate 'phrasing', for example, fall in this category. How is TEX*muse*, that builds characters vertically, going to deal with this?

The common feature of these 'horizontal elements' is that they are entirely defined by the last note involved. By 'remembering' which notes in which staves have to be included in the group (the beaming, the slur, etc.), TEX can postpone its typesetting to the last note. The METAFONT program for this last character can include the horizontal element, sticking out to the left. (This has in fact already been implemented for beams; slurs and other

[14] Is this not, after all, what happens with TEX itself? Most needs are already met, but in addition anybody could program his own, learning from existing examples, notoriously those of LATEX and its packages, and, of course, Plain TEX and *The TEXbook*. Look at me: I am a total amateur, I have never taken even a lesson in programming. But here I am, planning to teach music and METAFONT-programming to TEX.

elements are analogous. The 'remembering' takes place in METAFONT, which is much better at handling information.) Horizontal elements remain as such: they are not 'broken' into different pieces for different vertical agglomerates.

There are other elements that could imply a disturbance of the basic model of TEX*muse*. Notes are often 'adorned' with all kinds of symbols: the ♯ and ♭ signs to begin with, but also many different 'accents' (dots, lines, hats, ...), fingering indications, dynamic markings, etc. Many of them are actually vertically aligned with the notes, so that they can be included in the characters just as note-heads are — no special treatment is needed. But others go to the left or the right of the notes. The important implication of this is that these elements make spacing more complicated: in general, spacing is affected by them *only* if they generate collisions. In other words: if there is enough room for the element to attach to a note without colliding with the previous (or the next) one, it is simply appended; but if it doesn't fit in the available space, the note has to be moved (and, with it, all the notes in the same vertical axis).

TEX*muse* has to keep track of the width of the characters and the space between them. This provided, the task is trivial: the extra space needed can always be added to the right-most character of the potential collision. It will never be necessary to modify a previously built character (which was the potential threat to the whole system). This bears some relation to the matter of horizontal constructions treated above: there is actually no problem, because additions and adjustments can always be made in the 'current' character, without second thoughts about shipped-out notes.

9 Conclusion of the first part

The implementation of these two things — horizontal and offset elements — has involved a deep change in the nature of the whole system. But this was reserved for me to discover only when I started programming. We will see that many interesting things would happen in the process: I found unexpected and risible redundancies in my ideas (section 12.3), a particular problem that seemed relatively easy but is giving me trouble to this very day (section 14), and, most strikingly, a slip on the part of the Grand Wizard: a minor but unjustifiable omission in METAFONT, hard to believe coming from him. I shall have occasion to complain later (note 23).

What matters is that the phase of qualitative design was over: I had found a promising general idea and I had tested it with all that was available to me — all the challenges that had beaten other system and other ideas. TEX*muse* had survived, and it was now due to come into existence.

PART II
Toward and into implementation

In the original version of this article (written before any code whatsoever), this second part was devoted to a more detailed description of the planned workings of TEX*muse*, all in future tense. Discussed were the actual way TEX's glue would be applied, the handling of horizontal elements (beams, slurs, and the like), and the basic idea behind 'teaching music to TEX'. After that came an example, in which I played a computer running TEX*muse* on a hypothetical input, so as to illustrate the whole process.

Today, however, the actual activity of coding it all has revealed many 'flaws', to name them generically. They are all amusing. For example, influenced by Figure 2 above — and I don't know by what else — I was thinking that the TEXing of a TEX*muse* file would require three passes. I was even thinking about ways to let the user know that more passes were needed, and ways to 'fake' the final TEX box just as the `draft` option does for the graphicx LATEX package! Soon I was to realize that there was no need for all this...

The point is that in the process of coding I found these flaws, and the changes this has meant for the system are of far greater importance than many of the detailed descriptions of the original prospective account. This present second part will therefore omit many of those descriptions in favor of the latest developments. An original section on 'teaching music to TEX' stays pretty much the same; the section on spacing is still there, but essentially changed, since here the most important revision took place. Finally, I'm happy now to present a 'sample' instead of the old 'example' — now I *have* a computer running (part of) TEX*muse* on a real input. After all that, I present a totally new set of open questions, questions that were always present but have started to come to focus as their eventual implementation approaches.

10 Teaching music to TEX

The main consequence of the premise of horizontal output, as discussed in section 10, is that TEX has to understand the rhythmic characteristics of the user's input music. Thanks to that, it will be able to extract, from the input for each staff, the notes that need to be typeset in any given vertical axis (representing a 'moment' in the music).

Notes played at the same time have the same x coordinate (the different instruments in which they are played are represented by different y's).[15] All that TEX needs is to assign a value — the value of its x — to each note of each staff; then, it will sweep all these values, and build vertical characters with notes that have the same x. We are not yet dealing with 'material' dimensions, measurable, say, in millimeters. The actual horizontal placement of the notes in the staff is not a direct function of their x. So far the procedure gives TEX only the vertical correspondences between horizontal input.

A 'unit' has to be defined for this virtual axis. Since the rhythmically smallest note in usual practice is the 128^{th}-note (a note with 5 flags, 32 of which equal a 'crotchet' or quarter-note), a fourth of this value would be safe as a unit.[16] The quantum q is then defined as the duration of a 512^{th}-note. All other notes can be interpreted in terms of q. For example, the quarter note will be $64q$, the whole note will be $512q$.

Armed with these 'map of musical time', TEX reads the input a first time.[17] All the first notes in each staff occur at $0q$. From then on, according to the duration of the previous note, TEX finds the value at which all the notes occur. An example is the fragment in Figure 3. The first note (at $0q$) in the bottom staff is known to be a quarter-note (the user specifies this), and therefore the second note will be at $64q$. This second note is an eighth-note, so that the third one will come at $92q$. The count goes on, and then the procedure is done for the top staff, for which the values 0, 16, 32, 64, 72, 80, and 112 are found.

From this, TEX knows: there are notes in both staves for position $0q$; at $16q$ and $32q$, on the other hand, only the right hand has a note; and so on. The vertical correspondences are thus understood.

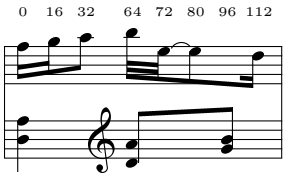
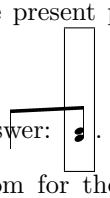
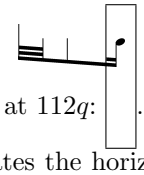


Figure 3: A fragment (m. 10 of Bartók's *Sonatina*) with its rhythmic 'quantization'.

11 Horizontal elements

Figure 3 features also instances of the most important kind of 'horizontal element': notes 2 and 3 in the bottom staff, or notes 1–3 and 4–7 in the top staff, are connected by 'beams'. How are beams created by TEX*muse*? A boxed insertion of the second-to-last character of Figure 3 disturbs the layout of the present paragraph, but it gives the key to the answer:  . This character is fairly tall — it makes room for the top staff, although there is nothing there (moment $96q$). But the important point is that it contains, sticking to the left, the *complete* beam that connects that note to the previous one (as well as the stems for both notes).

Since the stems of all the notes included in the beam are typeset when the beam itself is typeset, every note in a beam except the last one features only the note-head. For example, the character for $80q$ illustrates this (and also the procedure for musical 'ties', which is basically the same but simpler):  . This beam is completed only at $112q$:  .[18]

This is how METAFONT creates the horizontal elements. The other part of the problem is getting TEX to instruct METAFONT to do so. When TEX

[15] Here I am ignoring the dimension of 'pitch', and talking about the 'third dimension' of musical notation, as introduced on page 171.

[16] I have never seen a note of this kind in print, but it's good to provide for it. Besides, the smaller the unit, the easier and more precise will be irregular subdivisions (triplets, quintuplets, etc.), of frequent appearance in music.

[17] Here it is that a 'several-pass' system is implied: TEX reads the input a first time to deduce the correspondences, and a second time to actually build the characters according to those correspondences. In fact, the main programming mechanism for this is to define the same commands differently for each of the stages. Usual implementations of TEX today run METAFONT automatically when `tfm` or `pk` files are not present, so that the procedure does *not* require TEX to run twice on the file, as I first thought.

[18] You might be wondering about the different number of beams for different notes in the last illustration. This is another thing that depends on the rhythm of the notes involved, and this dependence is actually much more interesting and challenging to express algorithmically than the 'simple' matter of compiling the map of section 10. I have found an algorithm for this — a very nice one, I think, that uses METAFONT's weights and `undraw` technique — which I would like to present. Unfortunately, the mere explaining of the requirements would take so much space that it's hardly feasible.

On the other hand, a problem that I haven't solved is how to decide the angle of inclination of the beam. (In part, it's unsolved because I've been unable to find, explicitly stated, the very complex rules concerning this angle.) For the present examples I cheated, deciding the inclination on my own.

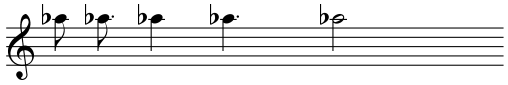


Figure 4: Different spacing after different note-values. (Excerpt from the violin part of Lustosławski's *Partita for Violin and Piano*.)

finds an opening '|' character in the user's input, the convention is that the following notes are to be part of a beam. So it suppresses the generation of stems, and starts building a list of the notes for the beam. When it finds the matching '|' (that closes the beam), the list is complete, and TEX passes it to METAFONT. Having kept track of exactly where all notes were placed, METAFONT is able to draw their stems and the connecting beam.

A paired '(' and ')' analogously indicates slurs (TEX starts a list of notes when it finds the first, and gives it to METAFONT when it finds the second); while '=' (as in Finale) indicates ties.

This is, in the main, how TEX*muse* generates horizontal elements from straightforward user instructions.

12 The spacing

Here we come to the heart of the matter. As noted before, the main motivation for this whole project was how to get spacing automatically without involving the user at all (unless he *wants* to be involved). It is finally time to put forward my solution.

Figure 4 will clarify — for the case of a single voice — what the goal is when talking about 'spacing'. The five notes involved are different from each other only in their rhythmic value, which increases from note to note: eighth-note, dotted-eighth-note, quarter-note, dotted-quarter-note, half-note. The spacing has to increase accordingly.[19]

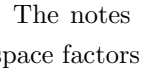
12.1 How to space the notes

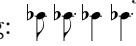
The first thing to do is to generalize the goal, rigorously, for the many-voices situation. How is a many-staff character to be treated, since it usually contains notes of different rhythmic values? For example, back in Figure 3, the first note contains a quarter-note and a 16th-note. Which one defines the spacing? (The answer might seem obvious, but we need to express it algorithmically).

Imagine the whole 'map' of musical time (as defined in section 10) collapsing into $y = 0$ — that is, the projection of the whole thing into the horizontal axis. (This projection, a series of dots scattered along a line, is a representation of what is known as the 'compound rhythm' of the entire ensemble.) Spacing is inferred from this projection. For every 'musical moment', TEX knows how long it will take for the next note to sound anywhere. Since between the first two notes of Figure 3 there are $16q$ from note 1 to note 2, TEX will treat the character as precisely that: as a $16q$ (a 16th-note).

But don't leap to conclusions too fast. Take what happens at $80q$: the next note comes in the bottom staff at 96, i.e., $16q$ afterwards. Therefore, in spite of being an eighth-note, this character will also be treated as a 16th-note ($16q$ long). It's not simply the shortest note present that determines spacing; it's the shortest note *in the compound rhythm.*[20]

12.2 Spacing them

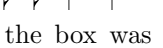
Now that we know how to treat the notes, spacing-wise, according to their rhythmical nature, let's have a closer look at how TEX actually does it in Figure 4. The notes are  , and TEX knows what space factors to apply to them: the respective values are 1000, 2000, 4000, 8000, and 16000. Thus, the spacing required to fill out the line — i.e., the amount of stretching of the inter-note spaces — will be proportionally distributed in 1 : 2 ratios.

This alone doesn't work. TEX's algorithm is designed to *minimize* stretching for text, so the difference between spaces will be hardly noticeable. In fact, this is what you would get from regular TEX spacing:  . The difference is there, but it's by far insufficient for a musician's needs. So you need to 'fool' TEX, telling it the box is much wider

[19] There are several models as to how the space is a function of the rhythmic value. One is called 'Fibonacci', and works by assigning to each note a space equal to 0.618 the space of the immediately longer note. I tend to go for this model; but Figure 4 is actually set with a binary model, whereby each note receives twice the space of the immediately shorter one. This seems to work better when, as in the example, there are dotted notes. Obviously, this parameter will be user-modifiable in TEX*muse*.

[20] Under the heading "The spacing of notes", the authors of MusiXTEX write that "it can lead to interesting algorithms." For them, however, "it is not an important point in practice." Why? Because, owing to the fact that sometimes the spacing is not that of the shortest note present — as happens at $80q$ in Figure 3 — "the typesetter has to take care of good readable spacings on his own." [8, pp. 6–7] OK, to me this is plain upside-down reasoning: 'since we're leaving this task to the user, the algorithm has no practical relevance.' My preference would be something like 'since it is totally impractical for the user to figure the spacing out, we should go and look for the algorithm.'

Devising the algorithm is not even the real difficulty. As we saw, it's immediately solved by inserting 'in the compound rhythm'. The hard part is how to *implement* the rule in TEX. The quantized map of musical time offers a solution.

than it actually is, so that it will stretch more dramatically. For example, telling TEX the box is twice as wide gives  .[21] In Figure 4 TEX was told the box was 2.5 times wider. The correct setting of this parameter, or more likely the dependence of this parameter on the music involved, is yet to be discovered.

Anyway, the same procedure took place in Figure 3 (this time the box was doubled in width), only applying space factors according to the compound rhythm.

12.3 But...

This looks great. It is as satisfactory an application of TEX's stretching algorithm as it can get. My main point is made.

And yet... it's redundant. For one thing, you'll have noticed that the in-text illustrations above lack a staff (the array of five lines). In the figures, the staff has been printed on top of METAFONT's characters with TEX's `\rule` commands. Since the beginning this was known to be a temporary solution. A professional system needs to provide for modifiable 'staff-ing', both vertically (more or less than 5 lines), and horizontally (a portion with 5 lines, followed by a portion without lines, followed by ...). The most reasonable solution is to attach the number of lines to each note: METAFONT will draw the necessary lines when drawing the note. But if the spacing is simply the stretching of blank spaces, who will draw the lines there?

More important, however, is that the drawing of horizontal elements involving many notes assumes, as explained in section 11, that METAFONT knows "exactly where those notes were placed". Otherwise it will be unable to draw slurs or beams that actually 'hit' the notes involved. I had concluded this already in 2002: "METAFONT needs a way to calculate *beforehand* the stretching that the construction of the line will perform. This amounts to implementing the relevant equations of the algorithm in a METAFONT function".

So I went ahead and did just that: I derived the equations (see appendix B), implemented them in METAFONT, and thus made it able to 'foresee' where TEX would finally place the notes. In this way, it knew how long to draw the staff lines, and where to find the notes under a beam.

OK, until one day something strange happens: I see that METAFONT's beams don't quite hit their notes. It took a lot of burdensome debugging for me to realize that, owing to a small mistake in the formulas, TEX and METAFONT were applying slightly different 'versions' of the algorithm. I even corrected my formulas diligently... Only a couple days later, I laughed when I noticed the real nonsense: the algorithm was being applied twice!

So today this has changed. It is METAFONT, not TEX, who applies stretching, according to formulas derived from TEX's stretching algorithm. The spacing is thus included in the font, and TEX limits itself to printing the characters out, one after another, without any space.[22]

12.4 Offset elements

The last aspect of spacing concerns elements that stick out to the left or to the right of their notes. Many musical elements do: an example is the treble clef 𝄞 in the middle of the bottom staff in Figure 3. It sticks out to the left of the next note, being separated from it 'rigidly'. If the note moves, the clef moves with it.

And, conversely, if the clef had to be moved, the note would move with it. That is how offset elements could affect spacing. In this case it does not, because the clef doesn't have to be moved — there's plenty of room for it. But sometimes an offset element will collide with something else, and then it has to be moved. Its note will also move, and, with it, all other notes on the same vertical axis.

This is implemented as follows: when METAFONT completes a note, it finds its right extreme r (with respect to the note's axis), and saves this value in memory. Then, when it is drawing the next note, it will calculate the left extreme l (of the current note, with respect to its axis).[23] As seen in the previous subsection, METAFONT will know the space s

[21] This example was achieved, in the present LATEX article, with `\makebox[2\width][s]`...

[22] This change actually raises a basic, qualitative question: in addition to the spacing of the notes, there are many other things whose handling can be done either by TEX or by METAFONT (for example, line breaking and justification). Deciding where to handle them amounts to deciding on the nature of TEX*muse*: is it a TEX package with a METAFONT underpinning, or a METAFONT package with TEX interface? Needless to say, the criteria for the decision are too subtle — it most likely will be an 'arbitrary' decision.

[23] Now, just how is METAFONT to find the right and left extremes of a character? When I faced this question, I went confidently to the index of *The METAFONTbook*: there had to be a function for this. Or at least one that would give me the width of a character. I was amazed to learn that METAFONT offers no '`widthof` ⟨*picture*⟩' function of any kind. The Grand Wizard introduces, in his 'Dirty tricks' appendix, an algorithm for METAFONT to find the extremes of a pen (this is necessary for some pen functions). The trick is truly dirty, and Knuth is justified if he takes pride on it. But does this not really hide an omission? I'm not the one to tell, but given the way pictures are internally represented in METAFONT, I would imagine that finding the extremes would be

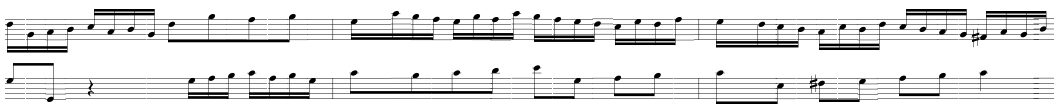


Figure 5: Mm. 2–4 of Bach's *Invention in C*, typeset by an incomplete TEX*muse* on Aug. 25th, 2004.

between these notes (more precisely between their axes) would be in the normal case, when no offset elements are involved and spacing is simply a result of rhythmic circumstances. It can then calculate whether this regular space will be enough to accommodate the notes (separated at least by a minimum 'framing space' f): is $s \geq r + l + f$? If it is, there's no problem; if it is not, the current note is moved to the right so that the collision will be avoided. Everything solved, the right extreme is calculated for the current note, and METAFONT is ready to go to the next.

'Moving the note to the right' means increasing its axis, which is shared by all other notes in the character. Since space can thus only *increase*, the adjustment can be done incrementally as offset elements in other notes are discovered to create additional collisions. The character is actually shipped out when all the notes have been added, all the collision tests have been made, and the axis has been moved as necessary.

13 A working sample

Figure 5 is a sample of what TEX*muse* is capable of (as of today, mid-2004). I've been actually developing the system with this sample — it's like a snapshot of TEX*muse*'s growth. The present article actually loads the current TEX*muse* version, and the input cited below is a verbatim copy of the commands used to generate Figure 5. I am willing to distribute the code to anyone interested.

Measures 2–4 of Bach's *Invention in C* were chosen for several reasons: rhythmic simplicity, one-voice-per-measure, no slurs or ties, no key- or time-signatures, no clefs. These things are not yet implemented; but none of them pose challenges, I simply haven't implemented these parts of the code (which are rather boring). *Challenging* things missing from the system will be mentioned in section 14.

an easy thing to program at the low level — just as METAFONT routinely calculates the 'total weight' of a picture, why should it not compute its extremes?

Knuth's roundabout method of finding the extremes of a pen can't be always applied, since a pen is a continuous and convex picture. Most pictures, and in particular musical characters, seldom fulfill those conditions. I had to develop a function from scratch (see note 25), but it turns out it's not right. A solution is yet to be found.

What I do want to show with this sample is that the system seems to be in fact possible. I would point mainly at the simplicity and naturality of the input that TEX*muse* demands from the user. Comparison is odious, and I shall refrain from citing the code that would be necessary in MusiXTEX to produce the three measures of Figure 5. But the point is important enough to ask MusiXTEX users to imagine it.

In TEX*muse*, all starts with the definition of the instruments involved. The system, as it stands today, rigidly assumes the top staff to be in treble-clef and the bottom one in bass-clef; of course, it will eventually provide for changes.

```
\newinstrument{righthand}
\newinstrument{lefthand}
```

Then, in the `texmuse` environment, actual music is input for each instrument. The initial string `DGAB`, for example, corresponds to the first four notes of the right-hand staff (whose standard names are, precisely, D, G, A, and B). Conventions are stated more fully in appendix A.

```
\begin{texmuse}
\meter44
\righthand{\rangefrom{G4}
    3|DGAB||CABG|\rangefrom{C5}|4DGFG|
    |3EAGF||EGFA||GFED||CEDF|
    \rangefrom{F4}|EDCB||ACBD||CBAG||\#FAGB|}
\lefthand{\rangefrom{G3}4|GG-|5R3R|GAB|
    |CABG||4CBCD||EG||AB|
    |CE-||\#F-G||AB|5C}
\end{texmuse}
```

Up to this point nothing has been typeset. The command `\musicbox` now tells TEX*muse* which instruments, and in what order, to build the fragment from. The user can thus use instruments flexibly, extract parts, re-order them, etc. There will also be additional musical-box commands, for example a `\musicparbox`, maybe a `\musicpage`, etc. That way, the layout is — for good — dissociated from the notes themselves.

In this case, we want a figure, so:

```
\begin{figure*}
\centering \musicbox{lefthand,righthand}
\caption{Mm.~2--4 of Bach's ..., 2004.}
\label{sample}
\end{figure*}
```

The result is in Figure 5. I'm satisfied with the way TEX is able to find all that it cares about, without making the user care about it too.

14 'The unanswered question'

In addition to obvious incompleteness, the sample shows signs of two particular problems that haven't been totally solved: the barlines, to begin with. Not only should they be drawn across the two staves (not just on them), but, if you look carefully, you'll notice that they have enlarged the spacing of the second note next to them. Barlines have proven a hard thing to implement, even qualitatively.[24]

The second problem can be seen in the last sharp-sign (♯): it features a spurious horizontal line. This is a result of METAFONT not having a direct way to find the extremes of its pictures (see note 23). To do that, the picture has to be made continuous, and therefore all 'additions' to the notes, such as offset elements like the sharp-sign, have to be joined to the note-head. The first ♯ also has this spurious line, only it's covered by the staff line. The horizontal line would of course be deleted after the extremes have been found — I haven't just coded that. The procedure strikes me as *ad hoc*, and I am definitely on the watch for better options.[25]

It was mentioned that TEX*muse* is not yet able to decide the angle of inclination for beams. There are other problems of this very interesting kind, that involve looking for (or deducing) the explicit list of requirements for good music typesetting: the most important ones are an algorithm to decide the order and placement of accidentals in a chord; a function for the ideal, context-dependent, ratio of space factor from rhythmic values to each other; a procedure to shift note-heads in chords with seconds; and the rigorous definition of the aspect of slurs and ties.

Another problem has to do with rhythms that are not binary: triplets, quintuplets, etc. There is an embryonic model for this, but it is as yet only qualitative: 'tuplets' can be treated as horizontal elements. Long pieces, on the other hand, will raise problems not yet dealt with, ranging from defining a concept of system-breaking (like a mixture of line-breaking and page-breaking procedures) to the good handling of space and justification of lines.

The most important issue is, however, keeping the design flexible. Up to this moment, only standard notation has been implemented — and there's still a long way to go with it. 'Programmability' is the only premise in which the sample shows no real, positive progress.

15 Conclusion

I hope this article has been interesting, or at least provocative. This report has to end here, but the project is only started. I look forward to any reaction from the TEX community, particularly of course from those interested in typesetting music with TEX. Just as happened with the first version of this article, the task of writing this has clarified many things to me. With the picture a little more clear now, I'm ready for a new session of actual coding. I'll keep in touch if anything interesting comes along.

References

[1] Ramón Casares. METATEX. *TUGboat*, 23(3/4):313–318, 2002.

[2] F. H. Gilson. *Music typography: Specimens of music types*. F. H. Gilson, Boston, 1885.

[3] Donald E. Knuth. *The TEXbook*. Addison Wesley, Reading, Mass., 1986.

[4] Donald E. Knuth. *The METAFONTbook*. Addison Wesley, Reading, Mass., 1986.

[5] Thomas E. Leathrum, Geoffrey Tobin, and Daniel H. Luecking. *Pictures in TEX with Metafont and MetaPost*. 2002. File `mfpicdoc.tex`, documentation to mfpic 0.6.

[6] David Salomon. *The Advanced TEXbook*. Springer Verlag, New York, 1995.

[7] Daniel Taupin. MusicTEX: Using TEX to write polyphonic or instrumental music. *TUGboat*, 14(3):212–220, 1993.

[8] Daniel Taupin, Ross Mitchell, and Andreas Egler. *MusiXTEX: Using TEX to write polyphonic or instrumental music*. 2001. File `musixdoc.tex`, documentation to MusiXTEX version T.98.

[24] There are in principle three ways of interpreting barlines in TEX*muse* terms: they could be treated as an extra note, 0q-long; they could be thought of as 'offset elements' to the left of the next note; or as 'offset elements' to the right of the previous one. Each of these approaches generates a series of side effects, surprising enough to make me unable to recall them all. The sample uses the second approach, and the repeated move of the note's axis confounds TEX*muse* about its real extremes.

This whole issue has led me to a recent rethinking of the whole process of drawing the characters. It's been decided that TEX (or METAFONT? — see note 22) 'orders' the elements of the notes, rather than following a first-come-first-drawn attitude: note-heads are drawn first, stems second, offset elements third, horizontal elements fourth, barlines fifth (or something like that). This ordering frightens me, though, because it might cut down on flexibility: what if the user wants something that cannot be easily classified and ordered?

[25] In fact, this solution is incomplete: there is no easy way of deleting precisely and only the line. I had imagined it would be done by `culling` appropriately (the line has a different weight from the signs), but it's much more complicated than that.

Appendices

A The user's input

The conventions for the input of the notes are designed to minimize the burden on the user's memory:

Rhythm is indicated through a convention standardized by Finale (and taken on by Sibelius): `5` means 'quarter-note' (crotchet); higher numbers are longer notes (`6`=half-note, `7`=whole note, etc.); and lower numbers are shorter notes (`4`=eighth-note, `3`=16^{th}-note, etc.). So, when TEX finds a `3`, it takes all following notes to be 16^{th}-notes, until a new number changes the value.

Pitch is given by upper-case letters following the usual note names: `A` through `G` (and optionally `H` for German users). Since there are many A's, many B's, etc., TEX needs a way to know which one the user means, which is achieved by `\rangefrom`. For example, `\rangefrom{C3}` means that the following notes are in the octave of the middle-C (which is in fact C3). If a note needs to be an octave higher or lower than set by `\rangefrom`, the modifiers `+` and `-` can be used (there's a `G-` in the left hand in the sample). `\rangefrom` can be used multiple times to set new ranges for different passages.

Rests are indicated by an 'R'. (In the sample, there's a quarter-rest, and there should be a 16^{th}-rest, but I haven't yet created the METAFONT picture for it.)

Beams are set, as mentioned, by enclosing the involved notes between two '`|`'s. TEX*muse* inserts beams appropriately according to rhythmic nature. Eventually, an optional argument to `|` will be available that allows the user to beam individual notes to each other, across measures, staves, etc.

Slurs are not yet implemented, but will be produced by enclosing the notes between '`(`' and '`)`'. As in Finale, a '`=`' will create a **tie**.

Accidentals and **accents**, being of frequent use, are taken care of by one-character, visually-suggestive commands (such as `\#` for the ♯s in the sample).

License is given to the user about things like extra, meaningless and unintended blank spaces (that are disastrous for MusiXTEX), or where to type the rhythmic-value numbers (in the sample there are both '`3|`' and '`|3`', but both work the same way).

B Formula for the horizontal position of a character

This appendix explains how the formula for the final horizontal position of a character has been found (because METAFONT needs to know it, see section 12.3).

The musical line consists of single-character words, separated by stretchable spaces. The position before the n^{th} character in the line is then the sum of the widths of all the previous characters and the spaces after each of them. These spaces are stretched according to TEX's glue rules given in [3, pp. 75ff.].

The particular use TEX*muse* makes of this algorithm implies the following assumptions:

- The line is always stretched.
- There is no infinite glue (`\hfil`, ...) in the line.
- The normal interword space and the extra space of the font are 0 (this to allow dealing with space factors less and greater than 2000 with no change in behavior).

Let l_i be the width ('length') of the i^{th} character of the line, and g_i the space (glue) added after it. It is then clear that the horizontal position for the n^{th} character is given by

$$\sum_{i=1}^{n-1}(l_i + g_i).$$

Now, g_i (the space after the i^{th} character) is a normal interword space plus the additional space due to stretching. But the normal interword space is 0, so g_i is limited to the stretching. If the line has a natural width of X, and a desired width of W, $W - X$ is the total amount of space added to it by stretching, this is, the total sum of g_i's in the line. Of this total stretchability, each character receives a portion according to its space factor f_i, thus:

$$\frac{g_i}{f_i} = \frac{\sum g_i}{\sum f_i} = \frac{W - X}{\sum f_i}$$

But X, the natural width of the line, is actually the natural width of all the characters, since the normal (non-stretched) space between them is 0. So, $X = \sum l_i$, and therefore

$$g_i = f_i \frac{W - \sum l_i}{\sum f_i}.$$

'Expanding' this gives the sought-for position for the n^{th} character (with z being the last character):

$$\sum_{i=1}^{n-1}\left[l_i + f_i \frac{W - \sum_{j=1}^{z} l_j}{\sum_{j=1}^{z} f_j}\right].$$

Note that the result does not depend on the stretchability of the font, and that because of the assumptions there is no need to invoke either the concept or the rigorous definition of *glue set ratio* r.

◇ Federico Garcia
Music Department
University of Pittsburgh
Pittsburgh, PA
feg8@pitt.edu

Font Forum

There is no end: Omega and Zapfino

William F. Adams

Abstract

The future of type is OpenType (Adobe and Microsoft's successor to Apple's "Royal" font technology which was licensed to Microsoft as TrueType), Unicode, and other extensions of TrueType and the Type 1 font format such as ATSUI (Apple Typographic System for Unicode Information). While TEX has been extended to support other new formats and standards such as .pdf, support for the new font formats has been limited at best.

Fortunately, for Unicode in TEX, we have Omega, which coupled with the other strengths of TEX, can be sufficient to take advantage of new technologies even without explicit support, by using the proper (or improper) techniques.

This paper will be an explanation and exploration of this, looking at a specific font and format (the `.dfont` ATSUI-enabled version of Zapfino), arguably very nearly a worst-case scenario, and how it can be dissassembled into individual glyphs and seamlessly stitched back together to automatically insert ligatures and swash and variant forms using ASCII markup in an otherwise ordinary `.tex` source file which can then be used in a prepress ready workflow.

Introduction

Apple's Mac OS X derives from NeXTstep, by way of OPENSTEP, with a grafting of Apple Macintosh user interface concepts. In terms of font support, it handles Mac Resource/Suitcase fonts (both Type 1 and TrueType) and PC TrueType fonts, but loses support for Unix `.pfa` NeXT style `.font` bundles. Apple's QuickDraw/GX is added as well, now known as ATSUI (Apple Typographic System for Unicode Information) or as AAT (Apple Advanced Typography) depending on the specific context or emphasis desired.

Mac OS X provides many of its system fonts in the new `.dfont` format, which, while a straightforward storing of a Mac-style TrueType font in the file proper (the datafork in Mac parlance) instead of the resource fork as was done with Mac OS 9 and earlier, is not equivalent to a PC format TrueType font stored in a `.ttf` file. Although there are now programs which can open and parse fonts stored in a `.dfont` (Pfaedit[1] is a notable example), my interpretation of Apple's licensing agreement leads me to believe that any such parsing or conversion would not be allowed by that license.

However, having purchased Mac OS X and its $10,000 worth of fonts, one cannot help but wish to use them. Although Zapfino works well in "Cocoa" programs in Mac OS X such as TextEdit.app, its special features such as ligatures are enabled by AAT which is unfortunately not supported by the more traditional "Carbon" Macintosh applications in which class all mainstream graphic design applications are, at this writing.[2] This is unfortunately quite limiting: either one must limit oneself to Cocoa applications, or in applications such as InDesign, make use of its Glyph palette to insert alternates and ligatures by repetitive pointing-and-clicking. Since there is no TEX variant which can access system fonts on Mac OS X as of this writing,[3] one must develop a work-around which allows one to access arbitrary fonts from within TEX and to simulate the sophisticated typesetting capabilities of OpenType or Apple Advanced Typography.

The large character sets of fonts such as Apple Chancery or Zapfino make accessing characters in 8-bit blocks untenable, so Omega is an obvious choice. This serves two purposes: first, it makes the typeface, Zapfino by Prof. Hermann Zapf available for use in TEX by way of Omega; second, it provides an encoding scheme and mechanism to access arbitrary ligatures and alternates which may be of use for other projects.

History

Zapfino had its origins in Prof. Zapf's 1944 sketchbook, when he was a mapping officer during World War II. A previous attempt to render those letterforms as type, Virtuosa Script for D. Stempel, had been rather compromised by the limitations of hot metal matrices, especially the swash letters. This design was revived when David Siegel in 1993, after working on the Euler project with Prof. Zapf, and after graduating approached him about a chaotic calligraphic typeface based upon an example done for the Society of Typographic Arts in Chicago. Remembering the page from his sketchbook, Prof. Zapf saw the chance for a design without compromises

[1] Renamed to *FontForge*, this wonderful program is available from `http://pfaedit.sourceforge.net`.

[2] Since then, the open source drawing program Cenon has been released for Mac OS X as well as OPENSTEP 4.2 and GNUstep. It is available from `http://www.cenon.info`. There are also SoftMagic's "Project-M", Stone Design's Create or Purgatory Design's Intaglio, but none are widely used.

[3] Jonathan Kew has since released XeTeX, a successor to his TEX/GX program for Apple's QuickDraw/GX, which runs on Mac OS X, thus making AAT fonts accessible. It is available from `http://scripts.sil.org/xetex`.

due to the advantages afforded by digital type technology.

Digitization was done by Gino Lee, but production halted due to personal problems, and languished until Prof. Zapf showed the design to Linotype. It was then rendered as a traditional, multiple alphabet typeface family.

Installation

This then begs the question of how does one install a font into a program (system) which doesn't have direct support for that font format or its capabilities? The solution is quite obvious in retrospect: consider what the system does support (PostScript by way of dvips and the `\special` mechanism) and where that intersects with the capabilities of systems which can use the font to its fullest (Encapsulated PostScript File graphics). The solution then is to load all of the characters of a font into a file so that they may then each be output as individual `.eps` files, stitch said files together as a virtual font and then rely on dvips to put everything back together.

Zapfino, however, has so many characters (1,417 in the version bundled with Mac OS X 10.2 "Jaguar"[4]) that Omega, with its support for Unicode which provides for large character sets, is needed. Omega also affords the Omega Translation Process (OTP), which is far more efficient at enabling long ligatures than the standard TEX or PostScript mechanisms. Fortunately, odvips supports the aforementioned `special` mechanism.

Although font metric information is probably not protectable, there is no reasonable method at present to access the data stored within the Zapfino font file which wouldn't run afoul of Apple's license forbidding decompilation or other modification. Presumably, however, a program using the nsText object could access such data on a per character basis and write that out in a useful format. In lieu of such, a copy of the `.afm` files provided by Volker Schaa (he had received a copy of the original Linotype Zapfino CD-ROM from Prof. Zapf as a gift) was used as a beginning point. These files were converted into standard `.tfm` files using afm2tfm and thence to `.pl` files using tftopl. The file for the font Zapfino One served as the basis for `zapfino.ovp`, the base font file. The utility `ovp2ovf` was then used to create `ovf` and `ofm` files for Omega to use. The files are stored in (as appropriate)

```
~/Library/texmf/fonts/ovp/apple/zapfino
~/Library/texmf/fonts/ovf/apple/zapfino
~/Library/texmf/fonts/ofm/apple/zapfino
```

Before testing could begin, it was necessary to have the letterforms themselves accessible to output. This was done by using Adobe InDesign to typeset an Adobe Tagged Text file which enumerated all of the characters in Zapfino. First, a single character was set in the font Zapfino in InDesign at 72 points size with 96 points leading and then exported (File | Export..., select "Adobe InDesign Tagged Text" in the Formats: pop-up), yielding a file with the following line needed for our purposes:

```
<cTypeface:><cSize:><cLeading:><cFont:><cHang:>
<pHyphenationLadderLimit:><pHyphenation:>
<pHyphenationZone:><pTabRuler:><ParaStyle:>
<pHyphenationLadderLimit:0><pHyphenation:0>
<pHyphenationZone:0.000000><pTabRuler:
28.000000\,Left\,.\,0\,\;56.000000\,Left\,.\,0
\,\;84.000000\,Left\,.\,0\,\;112.000000\,Left\,.
\,0\,\;140.000000\,Left\,.\,0\,\;168.000000\,
Left\,.\,0\,\;196.000000\,Left\,.\,0\,
\;216.000000\,Left\,.\,0\,\;224.000000\,Left\,.
\,0\,\;252.000000\,Left\,.\,0\,\;280.000000\,
Left\,.\,0\,\;308.000000\,Left\,.\,0\,
\;336.000000\,Left\,.\,0\,\;>
<cTypeface:Regular><cSize:72.000000>
<cLeading:96.000000><cFont:Zapfino>
<cHang:Baseline>A<0xFFFD><cTypeface:>
<cSize:><cLeading:><cFont:><cHang:>
<cSpecialGlyph:><cTypeface:Regular>
<cSize:72.000000><cLeading:96.000000>
<cFont:Zapfino><cHang:Baseline>
<cNextXChars:Page>
```

(The exported character was "A" — there is some additional text above and below said line, but it need merely be preserved in its entirety for later use.) After a little study and experimentation, it was found that the placed character could be replaced with `<cSpecialGlyph:####>` where `####` was a number ranging from 1 (the first character, "A" shown in the Unicode glyph palette in in Mac OS X) to 1417 (the last character, the open Apple symbol), so an Excel file was created with 1,417 rows and three columns. The first column was everything before the "A" endlessly repeated, with `<cSpecialGlyph:` added. The second column incremented the current row number starting from 1. The third column closed out the line, adding a `>` to close the `cSpecialGlyph` directive. This was exported as text and replaced the line shown above in the Adobe InDesign Tagged Text file which was then saved.

Next, a template file was created, as shown in Figure 1 (File | New | Document..., the Facing Pages checkbox cleared, the one for Master Text Frame

[4] Since this writing, Linotype has released Zapfino Extra OpenType which provides even more characters, most notably small caps, and Forte which provides five additional weights. The technique here should also work with this new version. More information on Zapfino Extra is available from `http://www.linotype.com/1897/linotypezapfinoextra-folder.html`

checked, the Page Size set to Letter, Orientation to Landscape, Margins and Columns left at their default). After clicking "OK" and getting a new document the "A-Master" page icon in the "Pages" palette (Window | Pages) was double-clicked to allow editing of the master text frame. The master text frame is then selected and set to the coordinates X: 43p6, Y: 31p6, W: 39p0 and H: 33p0 using the "Transform" palette (Window | Transform) as shown in Figure 2. The main document is then returned to by double-clicking on Page 1 in the Pages palette. Clicking with the Text tool in the text block, one then chooses File | Place... and navigates to the Adobe InDesign Tagged Text file created above and places it in the document so that it auto-flows to create 1,417 pages with one character per page (click on the Master Text Frame while holding down the `<Shift>` key). The file is then saved as `Zapfino-chars` in a convenient location.

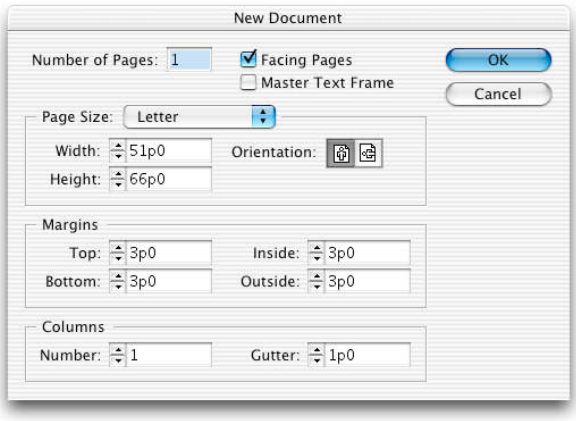


Figure 1: Adobe InDesign New Document Dialog

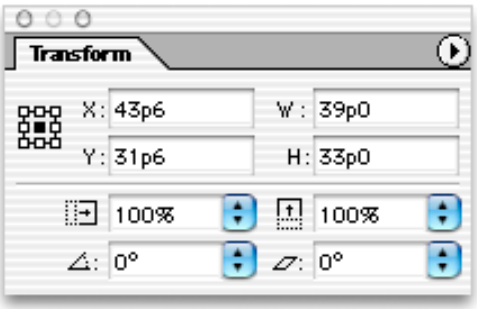


Figure 2: Adobe InDesign Transform Palette settings for master page text frame

Once we have this document with all of the desired characters, it is then a matter of exporting each

Figure 3: Z

page as a `.eps`. Fortunately, Adobe InDesign affords a menu option specifically for this, File | Export..., which includes direct support for the `.eps` format. Choosing "EPS" in the Formats pop-up menu takes one to a dialogue box where one can select various settings. For the initial font the settings used were: PostScript: Level 2, Color: Gray, Preview: None, Embed Fonts: Subset, Data Format: ASCII. InDesign is able to subset fonts in such a way that individual subsetted fonts may be recombined seamlessly within a PostScript file or `.pdf` without any of the encoding conflicts sometimes seen in fonts subsetted by Adobe Acrobat or other programs. The Data Format: must be set to ASCII, since (o)dvips cannot handle a binary encoded `.eps` file created in this fashion. The files are exported to `~/Library/texmf/fonts/eps/Apple/Zapfino` for later usage.

Then the `.pl` file for Zapfino-One was used as a basis for the initial `zapfino.ovp` Omega Virtual Font Property List. Notable settings which were necessary included setting the font's natural optical size `(DESIGNSIZE R 24)`. This technique is size-specific, and a different font must be made for each size which one wishes to typeset at. See the section *Peace* below for a work-around for this limitation.

With the character outlines now available, it is possible to place them within the virtual font using the `special` mechanism in odvips. Where each character has an entry like:

```
(MAP
   (SETCHAR 0 353)
   )
```

this is replaced with something like:

```
(MAP
   (SPECIAL PSfile=Zapfino-chars_277.eps)
   )
```

Unfortunately, when this is typeset the character is not positioned on the baseline, nor is it set to the correct size. Adjustments for the size and location of the `.eps` file placement must be worked out manually. Mostly a matter of trial and error, a test file

was created which placed a rule on the baseline, the test character and then another rule.

```
\font\zapfino=Zapfino at 24pt
\nopagenumbers
\overfullrule 0pt
\zapfino

\vrule depth 0pt \hskip-.5pt X

\vfill\eject\bye
```

Commands for controlling the position of the character and its size were then added to the `.pl` file and it was retypeset and adjustments were made until it was correct (see Figure 4). For example, the "IJ" entry looks like the following:

```
(MAP
    (PUSH)
    (MOVELEFT R 1.5734)
    (MOVEDOWN R 1.724)
    (SPECIAL PSfile=Zapfino-chars_277.eps
                  hscale=13.28 vscale=13.28)
    (POP)
    (MOVERIGHT R .543)
    )
```

First the current position is stored `(PUSH)`, then an adjustment is made for the offset of the character origin on the `.eps` file `(MOVELEFT)` and `(MOVEDOWN)`, the character is placed `(SPECIAL PSfile=Zapfino-chars_277.eps`, and scaled with `hscale` and `vscale`. Then the previous position is restored `(POP)` and the position advanced to match the `CHARWD (MOVERIGHT R .543)` (where .543 is the width of the "IJ" character).

With all of this done, one can begin testing the font so as to check the metrics of the characters. A number of the characters in Zapfino were re-drawn between the original Type 1 format and the version Apple bundled with Jaguar, so this step could not be skipped. How to space and test a new typeface design is well documented in several excellent references, most notably Stephen Moye's *Fontographer: Type by Design* and Walter Tracy's *Letters of Credit*, both of which are highly recommended to the aspiring type designer (or installer). In short, one sets various "standards" and other characters between them, adjusting the most common and easily spaced characters ("n" and "o") first, working toward those which are more difficult and assigning predetermined sidebearings to similar characters (so "m" gets the same sidebearings as "n"). Often when designing a new typeface, the initial attempt to set the sidebearings will result in a determination that certain characters must be re-drawn. Naturally that was not at issue here, and with the data gleaned from the `.afm` files, the metrics quickly reached a usable state. Much hastened in this case since the left sidebearings are preserved in the consistent placing of the characters on the page in the source file, so only the right sidebearings needed to be checked or adjusted.

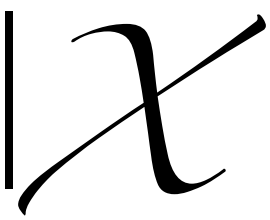


Figure 4: Final, correct `.eps` scaling and placement

Figure 5: TextEdit.app options for Zapfino in Mac OS X 10.2 were rather minimalistic

With the base character set available, next the ligatures and alternates needed to be provided for. Initially, Apple's options for Zapfino were somewhat limited as is evidenced by TextEdit.app (see Figure 5). WorldText.app, née GX/Write, exposed all of the capabilities encoded within the font, however (see Figure 6), and Apple has since expanded the nsText object to fully support Zapfino's myriad capa-

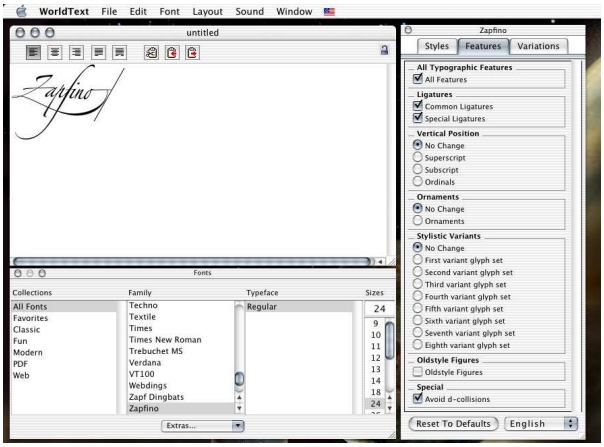


Figure 6: WorldText.app, provided in Apple's Developer Tools Samples folder, provides access to all of Zapfino's capabilities

bilities.[5] Although one may make various menu selections, or select characters from a Unicode "Character Palette" in Cocoa apps, or the Glyph palette in Adobe InDesign and other Adobe graphics applications, these options are not readily available in a text-stream composition-oriented tool like TEX.

Further, although OpenType and AAT are able to access characters by name, instead of directly with a number — and many of the more interesting characters in rich fonts such as Zapfino or Hoefler Text do *not* have Unicode code points — TEX and its variants (with the exception of the new XeTeX) require a number in an encoding vector for any given character. Toward this end, an encoding scheme was worked up to allow arbitrary ligatures of up to three characters in length, and to accommodate up to 32 variants for any given (unaccented character). While that last number may seem overkill, the OpenType specification provides for up to 20 variations of a character in its `salt` tag.[6] Having 16 bits available with Omega, the available bits were split into three sets, an initial set 6 bits long and two successive sets 5 bits long. The first set is long enough to encompass basic Latin capitals and minuscule (lowercase) letters as well as the numerals 0 through 9, with two bits left over. One of these was used as a "swash" bit, while the other remains available for use. The second and third sets encompass lowercase letters.

[5] See "Panther's Major Text Services Upgrade", `http://www.codepoetry.net/archives/2003/10/24/panthers_major_text_services_upgrade.php`

[6] See Tag: `'salt'` in `http://partners.adobe.com/asn/tech/type/opentype/appendices/features_pt.jsp`

Figure 7: Za

With all characters assigned to slots, it was then possible to create an Omega Translation Process (OTP) to replace characters with their appropriate ligatures:

```
input: 1;
output: 2;

states: VERBATIM;

expressions:

‘0’‘0’ => "{\zapfinoexpert " @"035A"}";
‘1’‘s’‘t’ => "{\zapfinoexpert " @"0653"}";
‘2’‘n’‘d’ => "{\zapfinoexpert " @"09A3"}";
‘3’‘r’‘d’ => "{\zapfinoexpert " @"0E23"}";
...
‘C’‘i’‘e’ => "{\zapfinoexpert " @"3504"}";
‘C’‘o’‘.’ => "{\zapfinoexpert " @"35DA"}";
‘D’‘r’‘.’ => "{\zapfinoexpert " @"3A3A"}";
...
‘T’‘h’ => "{\zapfinoexpert " @"78FA"}";
...
‘f’‘i’ => "{\zapfinoexpert " @"A91A"}";
%‘f’‘i’ => "{\zapfinoexpert " @"10DD"}";%fi-alt
%‘f’‘i’ => "{\zapfinoexpert " @"10DE"}";%fi-swash
...
‘g’‘g’ => "{\zapfinoexpert " @"ACDA"}";
%‘g’‘g’ => "{\zapfinoexpert " @"ACDB"}";%gg-alt
...
‘u’‘z’ => "{\zapfinoexpert " @"E73A"}";
‘w’‘n’ => "{\zapfinoexpert " @"EDBA"}";

@"F000 => <push: VERBATIM> ;

<VERBATIM>@"0021-@"007F => #(\1 + @"F000) ;

<VERBATIM>@"F001 => <pop:> ;

. => \1;
```

The Omega documentation[7] explains OTPs in detail. Each line in an OTP to handle a particular

[7] *Draft documentation for the Omega system* (John Plaice and Yannis Haralambous, 7 March 1998, available from `http://www.loria.fr/services/tex/moteurs/omega7mar1998.pdf`

case is fairly straightforward. First the characters to be replaced are identified (the `=>` indicates a replacement is being made), then a verbatim sequence of replacement commands is given within quotes, with a character's encoding being provided in an "escaped" fashion outside of the quotes (hence, `^^^^035a` is `"035A`). This was saved in a file `lat2zapf.otp` and processed with the command

```
otp2ocp lat2zapf.otp lat2zapf
```

and the files moved to `~/Library/texmf/omega/otp` and `~/Library/texmf/omega/ocp` respectively.

A number of font transformations normally handled with ligatures are transferred to the OTP. Although not shown, em and en dashes are created in the OTP, as are "smart" quotes. Similarly, the standard ligatures *ff*, *fi*, *fl* and *ffi* are handled in the OTP (there is no *ffl*). There are several reasons for this: conformance with other Omega fonts, to preclude ligature formation interfering with contextual processing of alternates, efficiency (OTPs can process a single string of arbitrary length, but TEX's ligature mechanism requires sequential processing of all iterations leading up to the final form; see below) and the small matter of Omega crashing when forming a ligature past the 8-bit point.

```
partial listing of cmr10.pl
(LIGTABLE
...
   (LABEL C f)
...
   (LIG C f O 13)
...
   (STOP)
   (LABEL O 13)
   (LIG C i O 16)
...
```

Note that there were several ligatures which were commented out with `%`, alternate or swash forms.

It was then possible to test the font for the first time with a (Plain) TEX file like:

```
\font\zapfino=Zapfino at 24pt
\font\zapfinoexpert=Zapfino-expert at 24pt
\ocp\zapfOCP=lat2zapf
\ocplist\zapfOCPlist=%
\addbeforeocplist 1 \zapfOCP
\nullocplist
\nopagenumbers
\overfullrule 0pt
\zapfino\pushocplist \zapfOCPlist

00 1st 2nd 3rd 4th 5th
   6th 7th 8th 9th 10th

Cie. Co. Dr. Esq. Ht. Jr. Ltd. McSullivan
Mlle. Mme. Mr. Mrs. Ms. No. 9 Sig. Sr.
Sra. Srta. St.

Thorn dicot draw presage fez type. affianced
effect fit fluent aft egg isthmus out
Zapf phone happy Arial art mesh spell
mass stay the matte spritz uzo own
\vfill\eject\bye
```

which shows all of the ligatures, as well as pointing out that perhaps using old-style figures as the default wasn't such a great idea, at least not if one intends to use the "0th" ordinal ligature.

Once the basic sidebearings of the font were set and the ligatures "wired up", the interplay of specific letterpairs needs to be addressed — the graphical interface which Mac OS X provides for Zapfino provides some hints on this, most notably, "Avoid *d* collisions." As one can see in figure 9, the standard, forward slanting lowercase *d* in Zapfino collides with the preceding character quite often, especially when preceded by a capital letter. Figure 8 also hints at awkward character interactions such as the near-intersection of *ppy*.

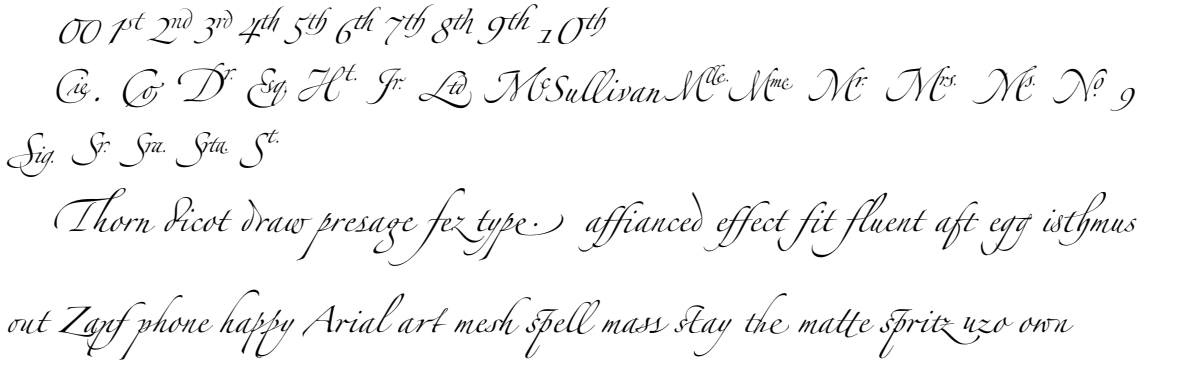


Figure 8: Initial test output

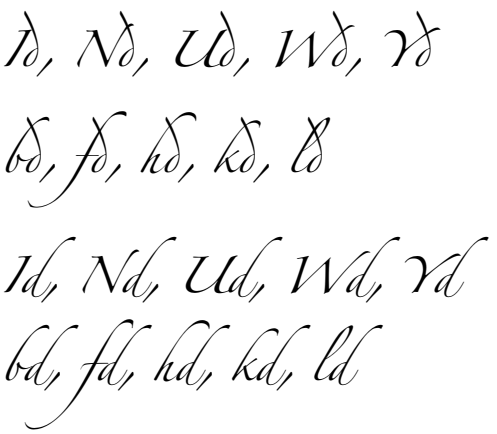


Figure 9: In Zapfino, *d*'s collide

While one is tempted to merely set the second style of *d* described in the font file, or Linotype's Adobe Type 1 version of Zapfino as `d.2`,[8] this brings to light an excellent chance to add some "chaos" to the typeface and pick different character versions.

The overall intent is to create a system which will make use of all possible characters in some circumstance, and a collection of test files which will test all such circumstances. While one is tempted to just "fix" all such occurrences, this leads to unnecessary processing effort. Instead, a listing of all possible digraphs (letterpairs) in the English language was created (drawing from *Webster's Dictionary*, *The Complete Works of Shakespeare* and *The King James Bible* — NeXT users will recognize the choice of references as those bundled with NeXTstep or readily available as texts for NeXT's Digital Librarian program), so that it was possible to determine that only *Id*, *Nd*, *Ud*, *Wd*, and *Yd* needed to be considered for the capitals (thus saving by not creating entries for *Fd*, *Hd*, *Jd*, *Kd*, *Td*, *Vd*, *Xd*, and *Zd* which do not occur in the English language sample set used). The following lines in the OTP prevent such collisions by replacing the normal *d* with the `d.2` alternate.

```
‘I’‘d’ => "I{\zapfinoexpert " @"FC62"}";
‘N’‘d’ => "N{\zapfinoexpert " @"FC62"}";
‘U’‘d’ => "U{\zapfinoexpert " @"FC62"}";
‘W’‘d’ => "W{\zapfinoexpert " @"FC62"}";
‘Y’‘d’ => "Y{\zapfinoexpert " @"FC62"}";
```

With the obvious change taken care of, more subtle changes were then considered. Working from the most frequently occurring letters to the rarer ones, a file for each letter, containing a word for each letterpair extant in the sample texts was typeset in the font Zapfino and then examined carefully for awkward interactions. When one was found, a test file was typeset with a basic macro to set the word with all possible variations for a given letter:

Figure 10: Zap

```
\def\testa#1#2#3{{#1}{#2}{#3}\par%   a
{#1}{\zapfinoexpert ^^^^fc02}{#3}\par%
{#1}{\zapfinoexpert ^^^^fc03}{#3}\par%
{#1}{\zapfinoexpert ^^^^fc04}{#3}\par%
{#1}{\zapfinoexpert ^^^^fc05}{#3}\par%
{#1}{\zapfinoexpert ^^^^fc06}{#3}\par%
}
```

allowing the selection of the best choice.

The normal usage was to set the text to be tested without any special treatment, and then to insert the macro into the text so as to call out alternates.

```
...
halfcock

\testf{hal}{f}{cock}
...
```

Twenty-six such macros were created, and a second set with only two arguments was also made for the capitals. Often it was necessary to test the characters before or after as well.

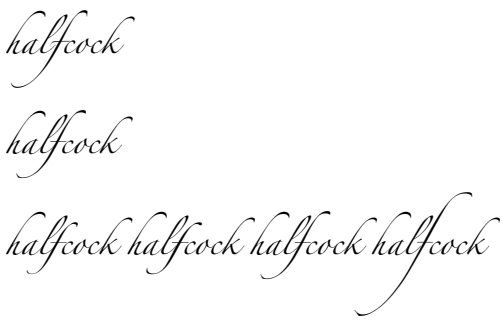


Figure 11: Test macro output

[8] Apparently, this route was taken by Apple and Linotype in the implementation in Mac OS X 10.3 "Panther".

Figure 12: Zapf without the *pf* ligature

Figure 14: Ligatures required examination of trigraphs and tetragraphs

The pair *nf* and a number of other pairs ending in *f* did not work well because the *f* in Zapfino One does not connect to the character before it, so a number of lines similar to the following were added to the OTP to handle such cases:

```
‘n’‘f’ => "n{\zapfinoexpert " @"FCA4"}";
‘o’‘f’ => "{\zapfinoexpert " @"FDC2 @"FCA4"}";
‘i’‘f’ => "i{\zapfinoexpert " @"FCA4"}";
‘i’‘j’ => "i{\zapfinoexpert " @"FD24"}";
```

Examining that file will show the reader which character pairs were felt to require adjustment. For the most part, replacing one character or the other (occasionally both) with an alternate provided an æsthetically pleasing combination, though a few uncommon pairs (*gj, Kz*) did not yield such.

Interestingly, the *gj* pair actually convinced me to revisit an initial decision to set the normal *j* (from the font Zapfino One) only at the end of characters, instead choosing it as the only appropriate form to use after a *g*.

Zapfino's many lowercase ligatures required that the list of digraphs be extended to encompass the trigraphs and tetragraphs which encompass these lettersets in addition to the digraphs mentioned above. For each such letter or ligature there is a file (e.g., `di-.txt` which is shown next), which encompasses the extant lettersets including said letter or ligature's letters with the hyphen indicating the location of the letters being checked for, so that file ranges from “Diagram” through “dizzy.” Since the *di* and *dr* ligatures do not reach as far to the left, most such letterpairs were acceptable. Notable exceptions were “ldr” and “ldi” which were set to make use of the alternate character `l.4` as shown in Figure 14.

```
Diagram diagram
Dibs dibs
Dice dice
Did did
Died died
Different different
Dig dig
Dihedron dihedron
Diiodemethane diiodemethane
Dijon Jehudijah
Dikdik dikdik
Dill dill
Dim dim
Din din
Diode diode
Dip dip
Diquat diquat
Dire dire
Distill distill
Ditto ditto
Diurnal diurnal
Divine divine
Diwali diwan
Dixie dixie
Diyarbakir
Dizzy dizzy
```

One hundred and fourteen such files were created. Usage with a leading capital is shown first if possible, followed by lower-case usage, again, if possible (*diy* apparently is only possible when capitalized as in *Diyarbakir*). If only lowercase usage is possible (not shown above) that line is indented by a space.

Zapfino is so narrow, however, with such a low x-height, and such large ascenders, that it is actually possible for the ascender on a *d* to collide with a character two characters before it as in, “led.” Thus a set of test files encompassing all characters with ascenders followed by any letter, followed by a “d” was created. Unfortunately, it is also possible for a

Figure 13: Alternate treatments

"d" at the beginning of a word to clash with the previous word if it ends with an ascender. Suggestions for dealing with this situation would be welcome.[9]

Doubled characters required special attention if there is no ligature for them, requiring that one check to determine which letter versions look attractive together.

After all such files were typeset and examined twice (once to establish the initial replacements, a second time to check for interactions between the replacements and characters which follow or precede them) it was possible to begin using Zapfino in Omega. The first such usage was for a holiday card as discussed in the section *Peace*.

Swash markup

That initial usage did not however, allow for user-level control of individual characters. One could of course use direct access with constructs such as `{\zapfinoexpert\char^^^^2804}`, but that's not exactly user-friendly (or even mnemonic). Although the use of swash alternates has been considered for Omega before,[10] and Alan Hoenig has considered macro-based schemes,[11] there exists no generally accepted system for indicating "swashness" in running text. The following version of a markup scheme for accessing arbitrary alternates and swashes is an experimental interface to determine how well this could be done using just an OTP.

The "swash" OTP uses the following options:

`+----a`	(first level alternate)
`+-----a`	(second)
`+------a`	(third)
`+-------a`	(fourth)
...	
`a----+`	(first level alternate at the end)
`a-----+`	(second level alternate)
&c.	

As always, the best laid plans ran into problems of implementation. The original intent was to add additional `+`s instead of continuing to add `-`s, but this became infeasible when a limit in the number of instructions (10,000) in a given state in an OTP was reached.[12]

[9] This is probably why Apple and Linotype chose to change the default "d" as noted in footnote number 8.

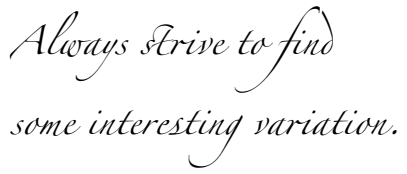
[10] Yannis Haralambous and John Plaice, "First applications of Ω: Adobe Poetica, Arabic, Greek, Khmer". *TUGboat* **15**(3), 1994.

[11] Alan Hoenig, "The Poetica family: Fancy fonts with TEX and LATEX". *TUGboat* **16**(3), 1995.

[12] My thanks to Giuseppe Bilotta for his patience and technical acumen, not merely determining the nature of the difficulty, but also providing a sample file showing how to work around it.

Figure 15: Zapf with the *pf* ligature

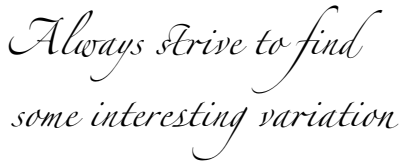
Before adding swash variations, here is a text using a modest set of alternates and most ligatures:



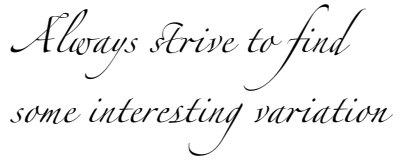
At the first level of ornamentation, which would be indicated as follows:

```
+-----Always strive to find-----+
+-----some interesting variation-----+
```

it appears thus:

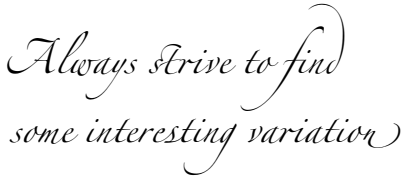


Continuing, the process yields:



or even:

Adding and subtracting hyphens, one might finally arrive at:



Left unaddressed is the matter of arbitrary swash variants in the middle of a text. Either of the above schemes can be used within a word, but interferes with spell-checking.

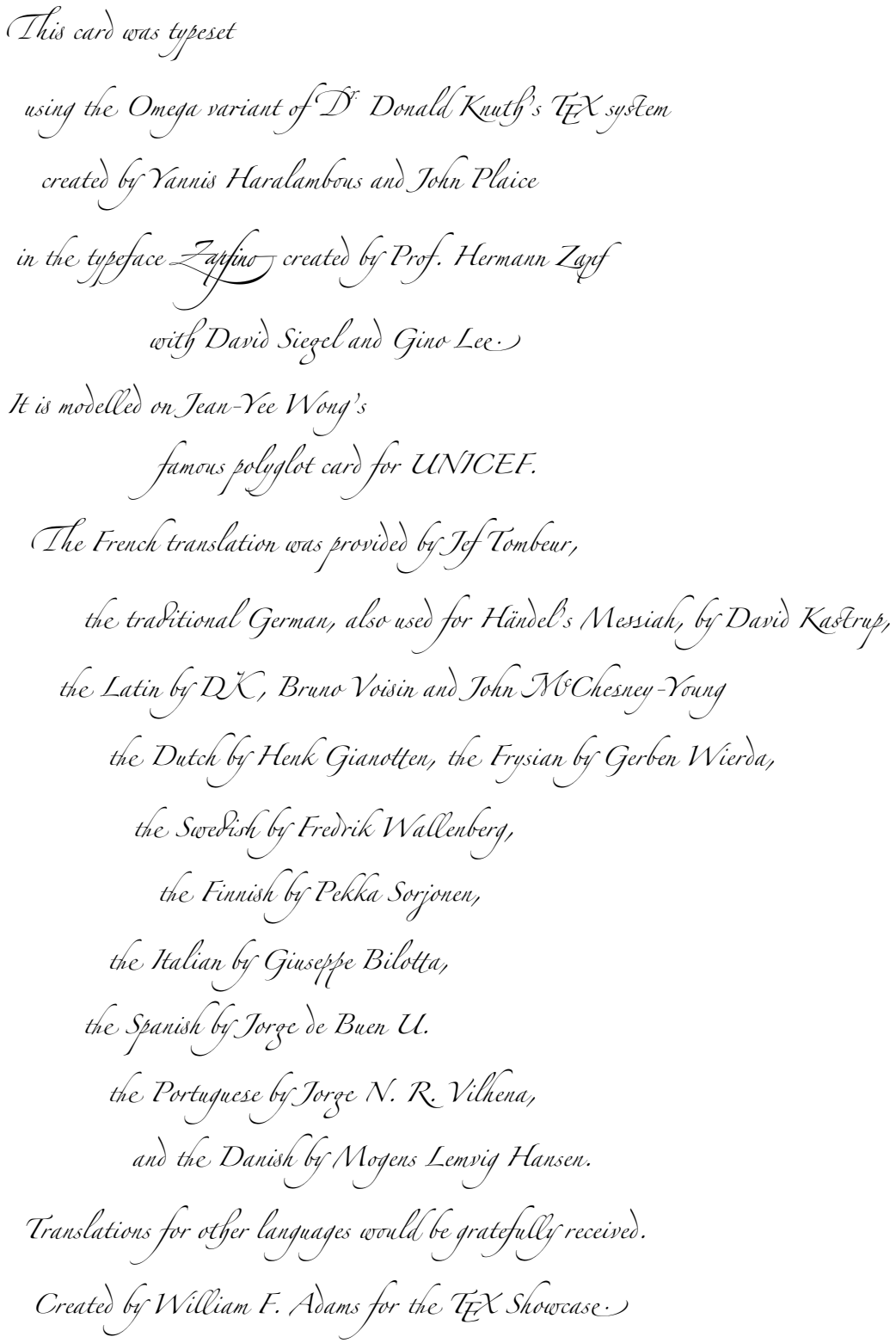


Figure 16: Peace on Earth card back page

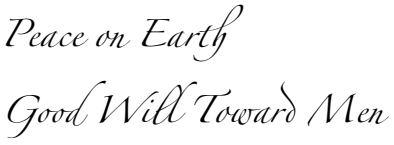


Figure 17: Peace on Earth interior

Figure 18: Zapfi

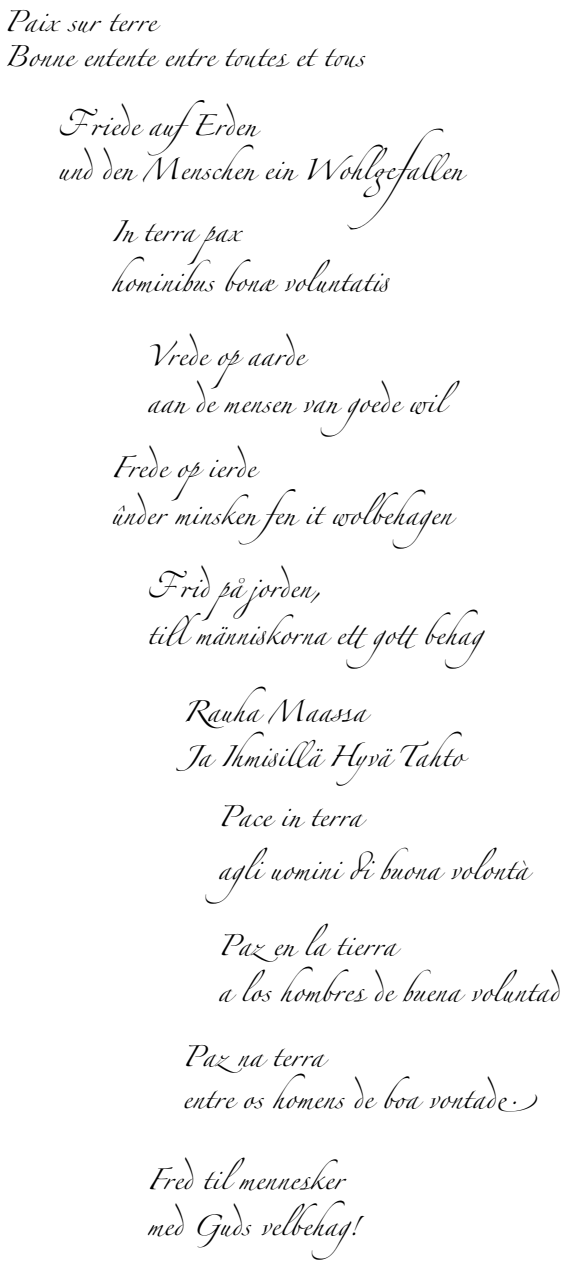


Figure 19: Peace on Earth card front page text

Peace

The first use of this Zapfino install in Omega was to set a holiday card modelled on Jeanyee Wong's famous polyglot card for UNICEF.[13] After this was initially set, it was announced on Usenet and various mailing lists related to TEX or typography in the hope of readers providing additional translations. Jef Tombeur and Apostolos Syropoulos were kind enough to provide translations for French and Greek respectively (though since Zapfino doesn't contain a full Greek alphabet the latter was provided typeset in the Kerkis font), and a revised version was made available as a part of the TEX Showcase.[14]

In the course of preparing this paper, a second, more successful, announcement and request for translations was made, with many people providing translations and commentary (I had neglected to mention the text, "Peace on earth, good will toward men" as being the latter part of the verse Luke 2:14 from *The King James Bible*, so in addition to straightforward transcriptions from various Bibles, I also received a number of personal interpretations — this verse is also quite controversial due to varying transcriptions of the original Greek text and subsequent emendations by various scribes) resulting in an expansion of the card to three pages, almost a dozen languages (Greek has been temporarily omitted) and twenty names.

Once typeset as a three page `.pdf`, the first and last pages (which were not filled completely) were cropped, then each page was exported as a `.eps` and imposed in Macromedia FreeHand where each was scaled appropriately to fit nicely on a tabloid or A3 sheet which could then be folded twice to make a gatefold card. This affords the illusion of having three different sizes of Zapfino available, despite only one font at a fixed size (24 pt.) having been made.

[13] Exhibit 214. UNICEF Christmas Card. Jeanyee Wong. 1962. *Two Thousand Years of Calligraphy: A Three-part exhibition organized by the Baltimore Museum of Art, the Peabody Institute Library and the Walters Art Gallery, June 6–July 18, 1965, A Comprehensive Catalog*: Baltimore, Maryland, 1965.

[14] `http://www.tug.org/texshowcase`

The Duck & the Mouse

by Jessica Adams

*One day in June a *DUCK* and a *MOUSE* were walking in a field when the *DUCK* saw a *TULIP*. The *DUCK* said, "Look at the pretty little red flower."*

*"Yes it's very pretty, but look..." said the *MOUSE*.*

*"What?" asked the *DUCK*.*

*"What is this funny long round thing?" asked the *MOUSE* digging in the leaves beneath the *TULIP*. "It's very pretty too."*

*"It's a people thing, a *PEN*." answered the *DUCK*.*

*"I wonder why it's here?" asked the *MOUSE* as it pulled the *PEN* out onto the path.*

*"Shhh! Listen!" whispered the *DUCK*.*

*"Humans! Hide!" hissed the *MOUSE*, and the two hid behind the *TULIP*.*

*"Don't worry, they're children." said the *DUCK*.*

*The big girl human asked, "Why'd you take my good *PEN*, Charlie?"*

"But Lizzie, I didn't mean to lose it!"

*"Look my *PEN*!" shouted the girl as she picked up her *PEN* from the ground where the *MOUSE* had dropped it.*

*The *DUCK* and the *MOUSE* breathed a sigh of relief and together they walked into the sunset.*

The End.

Figure 20: A child's story

```
\font\zapfino=Zapfino at 24pt
\font\zapfinoexpert=Zapfino-expert at 24pt
\ocp\zapfOCP=lat2zapf
\ocp\swashOCP=sws4zapf
%uncomment the rebusocp lines for pictures
%rebus\ocp\rebusOCP=rbs4zapf
\ocplist\zapfOCPlist=%
\addbeforeocplist 1 \zapfOCP
\addbeforeocplist 1 \swashOCP
%rebus\addbeforeocplist 1 \rebusOCP
\nullocplist
\nopagenumbers\overfullrule 0pt
\zapfino\pushocplist \zapfOCPlist
The +------D\kern2ptuck (-------ET{}
the +------Mouse \smallskip
by +------Jessica +-----Adams
\smallskip
One day in June a *DUCK* and a *MOUSE*
were walking in a field when the *DUCK*
saw a *TULIP*. The *DUCK* said,
``Look at the pretty little red flower.''

``Yes it's very pretty, but look...''
said the *MOUSE*.

``What?'' asked the *DUCK*.

``What is this funny long round thing?''
asked the *MOUSE*, digging in the leaves
beneath the *TULIP*. ``It's very pretty too.''

``It's a people thing, a *PEN*.''
answered the *DUCK*.

``I wonder why it's here?'' asked the *MOUSE*
as it pulled the *PEN* out onto the path.

``Shhh! Listen!'' whispered the *DUCK*.

``Humans! Hide!'' hissed the *MOUSE*,
and the two hid behind the *TULIP*.

``Don't worry, they're children.''
said the *DUCK*.

The big girl human asked,
''Why'd you take my good *PEN*, Charlie?"

``But Lizzie, I didn't mean to lose it!''

``Look, my *PEN*!'' shouted the girl
as she picked up her *PEN* from the ground
where the *MOUSE* had dropped it.

The *DUCK* and the *MOUSE*
breathed a sigh of relief and together they
walked into the sunset.

The End.\vfill\eject\bye
```

Zapfin

Figure 21: Zapfin

And pictures too

In addition to alternates and ligatures, Zapfino also has a number of ornaments available. Typically, access to such ornaments in TEX has been rather prosaically done using a macro which calls a particular ornament by its number. Although serviceable, this requires the user to have access to a chart which matches things up, and is not well-suited to usage in running text, or to easy typing or proofing. Oddly, although Helvetica seems to have a "Rebus" option in Mac OS X 10.3 "Panther", this does not seem to have been implemented for Zapfino, reducing one to point-and-clicking to access them from a glyph palette of some sort. Since that's not an option for normal TEX usage, an alternative scheme needed to be worked up.

Drawing upon the "Rebus" option for inspiration, a descriptive text was matched up for each ornament, and a new OTP was created to allow said text, set in all caps and surrounded by asterisks (e.g., `*PEN*`) to be replaced by the appropriate ornament when the OTP is loaded.

I prevailed upon my daughter to write a story making use of animals and things represented as ornaments in the Zapfino font. A typical setting of the story for proofreading purposes is shown on the facing page. The source text is shown to the left.

Some of the ornaments available with Zapfino include:

Figure 22: Ornaments

Lastly, the story with appropriate words replaced with ornaments from Zapfino (a rebus text) is shown on the following page in Figure 23.

Unfortunately, as noted above, the ornaments are numbered, not given descriptive labels. While some of them are obvious, others are less so. Suggestions on tags for unnamed ornaments would be welcome.

The Duck & the Mouse

by Jessica Adams

One day in June a 🦆 and a 🐭 were walking in a field when the 🦆 saw a 🌹. The 🦆 said, "Look at the pretty little red flower."

"Yes it's very pretty, but look..." said the 🐭.

"What?" asked the 🦆.

"What is this funny long round thing?" asked the 🐭 digging in the leaves beneath the 🌹. "It's very pretty too."

"It's a people thing, a 🍎." answered the 🦆.

"I wonder why it's here?" asked the 🐭 as it pulled the 🍎 out onto the path.

"Shhh! Listen!" whispered the 🦆.

"Humans! Hide!" hissed the 🐭, and the two hid behind the 🌹.

"Don't worry, they're children." said the 🦆.

The big girl human asked, "Why'd you take my good 🍎, Charlie?"

"But Lizzie, I didn't mean to lose it!"

"Look my 🍎!" shouted the girl as she picked up her 🍎 from the ground where the 🐭 had dropped it.

The 🦆 and the 🐭 breathed a sigh of relief and together they walked into the sunset.

The End.

Figure 23: A child's picture story

Figure 24: Zapfino without full ligation

Figure 25: Zapfino

Overview

Thus is it shown that with a complete disregard for efficiency, storage space and processing time, one can access *any* font from within TEX so long as it is supported on a platform whose capabilities somewhere intersect those utilities and variations which have grown up around TEX. This technique is completely platform agnostic after the initial stage, and if one foregoes the best font handling, can be done with bundled and free software, without recourse to commercial applications such as Adobe Acrobat.

Take an application which can access all of the characters in the OpenType or AAT/ATSUI font to typeset every character one wants to get at as a document, convert each page into `.eps` files, then create a virtual font which places the appropriate `.eps` file for a given letter, which is then used to typeset a PostScript file which places each character as necessary. When the resulting file is distilled with Adobe Acrobat Distiller the subsetted fonts in the `.eps` files are transparently stitched back together in a fashion which any reasonably capable PostScript interpreter or PDF viewing program can handle.

Advantages The good things about this technique include:

- No font conversion necessary
- No issues with licensing (Apple's software license forbids derivative products, so one may not convert or decompile typefaces covered by Apple's licensing agreements — Apple has even been stripping out tables to cripple their fonts so they won't work on other platforms if conversion is attempted)
- One can get at all 1,400+ characters in Zapfino without re-encoding or using awkward macros or active characters.
- Version invariant — already there have been *four* distinct versions of Zapfino, and little provision for backwards compatibility has been made.
- One is not limited to the normal TEX color models, but may use anything which InDesign has access to, including spot colors.

Disadvantages As alluded to above, this technique does have a number of drawbacks:

- Must make and store the individual `.eps` files (hundreds of megabytes, requires commercial software for the best results).
- Large file sizes (tens of megabytes for a single lengthy sentence — an attempt to create a complete sample of all digraphs and trigraphs ran into the Windows 9x file size limit of 512MB).
- Processing time (distill times in the tens of seconds for a couple of sentences, generating the `.ps` file with odvips takes even longer).

Future developments The following things remain to be done for this particular technique for accessing Zapfino:

- The base font needs to be filled in beyond just ISO Latin 1 so as to provide the balance of the Unicode-encoded characters available in Zapfino. This would be much easier if utilities such as afm2tfm and fontinst would completely support character sets larger than 8 bits.
- Similarly, most accented letters have swash variants, but the encoding scheme worked up for handling swashes and ligatures doesn't encompass such. Suggestions for a solution for this would be welcome.
- The text test set should be enlarged to encompass all possible trigraphs or even tetragraphs, and the OTP adjusted to at once increase randomness, and also reduce the incidence of certain characters which have a stronger presence than might be desireable.
- A LATEX 2_ε package should be put together to move usage out of the realm of hackery and into more mainstream production.
- Examine usage with Mac OS X Panther.

The *X* Factor

Since this presentation was made, Jonathan Kew has released XeTeX, the successor to TEX/GX, a TEX-variant which made use of QuickDraw/GX, from which Apple Advanced Typography is derived. Rapidly improving, it is well suited to personal use, or production use in an up-to-date production environment which is not hobbled by a need for backwards compatibility with older PostScript RIPs or other software.

It is especially well suited for use with languages which use character sets other than those based on Latin (which is its main focus).

By way of comparison, here is the source code for the Apple acknowledgement/"Thank You" which appeared in the TUG 2003 proceedings:

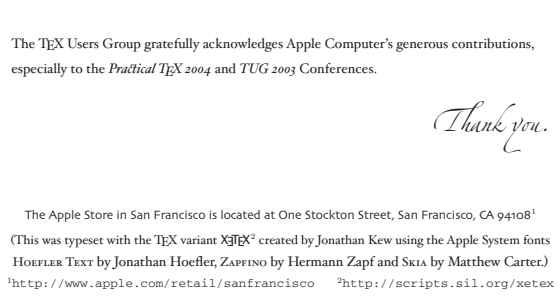


Figure 26: Thanking Apple

```
%&personaltex
%
%Copied from the XeTeX example file ‘‘XeTeX-notes.tex’’
\font\tx="Hoefler Text:Number Case=Upper Case Numbers" at 12pt
\font\txsmall="Hoefler Text:Number Case=Upper Case Numbers" at 11pt
\font\sc="Hoefler Text:Letter Case=Small Caps" at 11pt
\font\it="Hoefler Text Italic" at 12pt
\font\mt="Hoefler Text Black" at 16pt
\font\sh="Hoefler Text Black" at 12pt

\font\tt="Courier" at 11pt

\font\lg="Lucida Grande" at 11pt
\font\lgb="Lucida Grande Bold" at 16pt
\def\LaTeX{\leavevmode L\kern-.25em\raise.5ex\hbox{\sc a}\kern-.1em\TeX}
\def\XeTeX{\leavevmode
  \setbox0=\hbox{\lg X\lower.5ex\hbox{\kern-.1667em }\kern-.15em \TeX}%
  \dp0=0pt\ht0=0pt\box0 }
\def\mtXeTeX{\leavevmode
  \setbox0=\hbox{\lgb X\lower.5ex\hbox{\kern-.1667em }\kern-.15em \TeX}%
  \dp0=0pt\ht0=0pt\box0 }
\def\TeXgX{\TeX\lower.5ex\hbox{\kern-.15em G}\kern-.25em X}

% don't like how the hyphen in Hoefler appears, so we hack it:
\catcode‘\-=\active \def-{\lower.2ex\hbox{\char‘\-}\penalty0 }

%Copied from the XeTeX example file ‘‘story-zapfino.tex’’
\font\zapfinoreg="Zapfino" at 14pt
\font\zapfinofirst="Zapfino:Stylistic Variants=First variant glyph set" at 14pt
\font\zapfinosecond="Zapfino:Stylistic Variants=Second variant glyph set" at 14pt

%Copied from the XeTeX example file ‘‘FontSamples.tex’’
\frenchspacing % often works better with XeTeX
\baselineskip16pt
\parindent0pt
\parskip\medskipamount
\nopagenumbers

\font\1="Hoefler Text Italic:Letter Case=Small Caps" at 12pt
\font\1="Hoefler Text Italic" at 12pt
```

A note about Figures 3, 7, 10, 12 15, 18, 21, 24, 25 and 27

After my presentation in Hawai'i, Barbara Beeton specifically asked that I include an animation of the sequence of typing Zapfino, the contextual replacement of letters with the appropriate ligatures in the on-line version. Due to time constraints, I did not manage to do so. However, creating animated `.gif`s has always reminded me of study hall in my younger school days and doodling little figures in the margins of notebooks and pads, and of a Warner-Brothers book I had as a child which featured Wile E. Coyote and the Roadrunner of cartoon fame with such a sequence in its margin. I believe this is the first *TUGboat* article to include a flipbook (the figures in the upper right-hand corner of the recto pages). If present, a matching figure on the upper left of a page shows one what the text looks like without ligatures.

Figure 27: Zapfino ornament

◇ William F. Adams
ATLIS Graphics
75 Utley Drive, Suite 110
Camp Hill, PA 17011
USA
willadams@aol.com

Of the making of books, there is no end.

— The Prophet Muhammad

```
\font\italt="Hoefler Text Italic:Ligatures=Rare Ligatures,Diphthongs;
               Smart Swashes=Archaic Long s Swash;
               Character Alternatives=Swash Caps"
               at 12pt
\font\sr="Skia Regular" at 11pt
\font\1="Skia Regular:width=1.25" at 12pt
\font\1="Skia Regular:width=0.8" at 12pt
\font\1="Skia Regular:width=0.8;weight=1.5" at 12pt
\font\1="Skia Regular:width=0.8;weight=2.5;Number Case=Upper Case Numbers" at 12pt
\font\1="Skia Regular:width=0.8;weight=0.6;Ligatures=Rare Ligatures" at 12pt

\font\1="Skia Regular" at 18pt

\tx%
The \TeX\ Users Group gratefully acknowledges Apple Computer's generous contributions,

especially to the {\italt Practical \TeX\ 2004}
and {\italt TUG 2003} Conferences.
\bigskip
\hfill{\zapfinosecond Thank you.}
\bigskip \strut \bigskip
\centerline{\sr The Apple Store in San Francisco is located at One Stockton Street,
San Francisco, CA 94108$^1$}

\medskip
\centerline{\txsmall (This was typeset with the \TeX\ variant \XeTeX$^2$ created by Jonathan Kew
using the Apple System fonts}
\smallskip
\centerline{{\sc Hoefler Text} by Jonathan Hoefler, {\sc Zapfino} by %Prof.
Hermann Zapf and {\sc Skia} by Matthew Carter.)}
\smallskip
\centerline{$^1$\tt http://www.apple.com/retail/sanfrancisco ~ $^2$http://scripts.sil.org/xetex}

\vfill\eject\bye
```

The METAFONT approach: Implicit, relative, and analytical font design

Timothy Hall

Introduction

A past article in *Time*® magazine's *On-Line* monthly "submagazine" explored the world of do-it-yourself font creation and manipulation. The orientation of the article was to help a relative novice choose the right tools and techniques for whatever kind of font work was desired. The article was heavy on facts concerning a four-step process that might be familiar to readers of *TUGboat*:

1. Scan in a hand-drawn or copied glyph to form the basis for a character in your font.
2. Import the scanned image into a graphics software package, such as Illustrator, Fontographer, or CorelDraw.
3. Use the functionality of the package to trace the outline of the glyph.
4. Export the captured outline to an ASCII file that contains coded concatenated path segments.

In METAFONT syntax, the coded path segments will usually look something like this for a curved path:

$$z_0 \texttt{ .. controls } u_1 \texttt{ and } v_1 \texttt{ .. } z_1$$

or like this for straight lines:

$$z_0 \texttt{ -- } z_1 \texttt{ -- } \cdots \texttt{ -- } z_8$$

or perhaps combinations of these two forms. If each of these segments is named in a path array (**path** *pth[]*;), e.g., the third segment would be

$$\texttt{pth[3]=}z_2 \texttt{ .. controls } u_3 \texttt{ and } v_3 \texttt{ .. } z_3$$

then these path segments may be concatenated by the METAFONT operator "&" to form a cycle returning to the first point, such as this:

```
pth[0]=pth[1] & pth[2] & cycle
```

Finally, this path is typically filled to form the definition of the glyph:

```
fill pth[0];
```

As an example of this methodology, consider the following code for an arbitrary "mystery" character:*

```
numeric u; u = 1pt;
designsize := 10;
beginchar(99, 4.092pt#,
              5.80801pt#,
              0.264pt#); "myst";
fill  (0.594u,-0.264u)
 -- (0.594u,0.957u)
  & (0.594u,0.957u)
 .. controls (1.716u,2.046u)
    and (2.211u,2.805u)
     .. (2.31u,3.366u)
  & (2.31u,3.366u)
 .. controls (2.508u,3.927u)
    and (2.409u,4.686u)
     .. (2.013u,4.851u)
 -- (1.617u,4.917u)
 -- (0.693u,4.917u)
 -- (0.693u,5.808u)
 -- (1.881u,5.808u)
 .. controls (2.805u,5.808u)
    and (3.564u,5.412u)
     .. (3.399u,2.508u)
  & (3.399u,2.508u)
 .. controls (3.531u,1.32u)
    and (3.564u,0.693u)
     .. (3.696u,0u)
 -- (2.673u,0u)
  & (2.673u,0u)
 .. controls (2.574u,0.726u)
    and (2.475u,1.353u)
     .. (2.442u,1.782u)
  & (2.442u,1.782u)
 .. controls (1.881u,0.957u)
    and (1.221u,0.198u)
     .. (0.594u,-0.264u)
 -- cycle ;
cull currentpicture dropping (0,0);
endchar;
```

This form of code is perfectly acceptable to METAFONT, and this character definition (utilizing several unlisted parameter defaults) produces the glyph in Figure 1 (under `\mag=8`).

® *Time* is a registered trademark of AOL Time Warner, Inc.

* The material found herein does not depend on the particulars of any glyph definition.

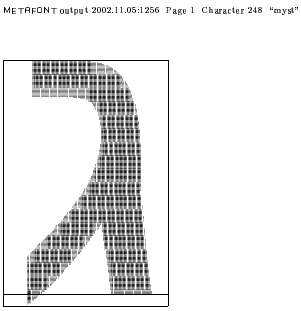


Figure 1: The original

However, there are several things "wrong" with the code in lines 6–36. METAFONT approaches font development from an implicit, relative, and analytical point of view, as compared to the explicit, absolute, and algorithmic calculations in Illustrator, Fontographer, and CorelDraw (among many others). In these other packages, Bézier curves are described in terms of control points, which are explicitly calculated within the software based on the absolute positions of critical points as parsed by the software. Illustrator, for one, when automatically tracing the outline of a glyph, decides where these critical points are, based on criteria and principles hidden from the user.

METAFONT, on the contrary, demands that the user decide where the critical points are located based on criteria and principles left to the designer, and expects the user to supply relative positions based on implicit relationships between the critical points. Only then will METAFONT calculate the control points for the glyph subpaths, and proceed to generate bitmaps for the image based on those calculations and numerous user-supplied parameter values.

In essence, simply stating a set of control points that blindly point to the outline of a glyph is not font design, and approaching font development this way is certainly not worthy of the amazing power and flexibility of METAFONT.

There is another thing "wrong" with the code in lines 3–5. The width, height, and depth of the character are stated as floating point numbers, one of which is given to five decimal places! This is definitely the work of a machine, and not a font designer. The control points listed in lines 6–36 also suffer from this feature. If the image had been held in a slightly different position when it was scanned using Illustrator, or a user had twitched slightly differently in positioning a serif in Fontographer, or a different version of the graphics software had been used with CorelDraw, or different hardware had been used in generating the numbers in lines 3–5, then slightly different numbers would be found there (certainly in the fifth decimal place).

This would not happen if the code development was under the control of a font designer who applied "the METAFONT approach." No amount of alignment issues, mouse sentitivity settings, version control aspects, or platform dependencies will ever affect the implicit, relative, and analytical relationships between critical points in the description of a font glyph, so none of these should be present in a METAFONT description of the glyph.

The METAFONT approach

Suppose you were interested in designing a font with hundreds of characters that followed the look and feel of the mystery character given in lines 1–38. You could scan and trace all of your images, and hope that the variations introduced by this process do not get in the way of your font's consistent look — or you might approach this task in the METAFONT way.

There are three straightforward transformations we may apply to lines 1–38 (and any others like them) to help reveal the underlying design structure of the mystery character, and help us apply it to all other characters in our desired font. This will go a long way towards ensuring a consistent look and feel to the entire font, and not simply be something applied to characters one by one. It is also possible that these transformations will reveal the absence of a consistent look and feel in the entire character set, thereby helping us modify lines 1–38 into a better glyph.

Implicit. The first transformation applies to all lines that contain explicit point definitions. If we label the first subpath as follows:

```
z1=(0.594u,-0.264u)
z2=(0.594u,0.957u)
z3=(2.31u,3.366u)
```

then we have:

```
x1=x2=0.594u; x3=2.31u;
y1+0.264u=y2-0.957u=0; y3=3.366u;
```

The resulting subpath would be:

z1--z2..tension α_2 and β_2..z3

The control points will be implicitly supplied by the **tension** statements during a later transformation.

The next subpath would be transformed as follows:

```
z4=(2.013u,4.851u)
z5=(1.617u,4.917u)
z6=(0.693u,4.917u)
z7=(0.693u,5.808u)
z8=(1.881u,5.808u)
```

... which produces:

```
x3=x4+0.297u=2.31u;
y3=y4-1.485u=3.366u;
z5=(1.617u,4.917u);
z6-z5=(-0.924u,0);
z7-z6=(0,0.891u);
z8-z7=(1.188u,0);
```

The resulting subpath would be:

`z3..tension` α_3 `and` β_3`..z4 --z5--z6--z7--z8`

Continuing this process, we arrive at the implicit description of the glyph, excerpted here.

```
55. numeric u; u = 1pt;
56. path pth[];
57. designsize := 10;
58. beginchar(99,4.092pt#,
59.               5.80801pt#,
60.               0.264pt#);"myst";
61. x1=x2=0.594u;
62. x3=x4+0.297u=2.31u;
63. x9+0.297u=x10=x11+1.023u=3.696u;
64. x12+0.231u=x11; x12-x13=1.848u;
65.
66. y1+0.264u=y2-0.957u=0;
67. y3=y4-1.485u=3.366u;
68. y10=y11=y9-2.508u=y12-1.782u=0;
69. y12-y13=2.046u;
70.
71.   [...]
72.
73. pth[1]=z1--z2;
74. pth[2]=z2..
75.        controls(1.716u,2.046u)
76.             and(2.211u,2.805u)
77.        ..z3;
78. pth[3]=z3..
79.        controls(2.508u,3.927u)
80.             and(2.409u,4.686u)
81.        ..z4--z5--z6--z7--
82.        z8..
83.        controls(2.805u,5.808u)
84.             and(3.564u,5.412u)
85.        ..z9;
86.
87.   [...]
88.
89. fill pth[1]
90.      for i=2upto6: & pth[i] endfor
91.      --cycle;
92. cull currentpicture dropping(0,0);
93. endchar;
```

The choice of points assigned to a given subpath is arbitrary, as long as the order of progression is retained. Indeed, the glyph produced by lines 55–93 looks exactly like that produced by lines 1–38 (see Figure 1 again), and would remain so should, for example, `pth[3]` be split into two subpaths each associated with a set of control points.

Relative. The next transformation enables the relative advantages of METAFONT design: Stating all points in terms of positions that are relative to the height, width, and depth of the character. In this way, you don't have to recalculate the points `z1`, `z2`, etc., every time you rescale the character to a different design size. Indeed, enabling the incredible variety of font characteristics found in the Computer Modern family is only possible when relative design policies are utilized.

The first question to answer in relative font design is, "on which dimension should the font be based?" If the font is to be monospaced, then standardization on the width would be appropriate. This would mean replacing the dimensions in lines 58–60 with

```
ds*pt#,
ds*(0.580801/0.4092)*pt#,
(0.264/0.4092)*pt#
```

in their respective positions, where `ds` is the design size. If the font is to have a uniform height, then the replacement dimensions would be

```
ds*(0.4092/0.580801)*pt#,
ds*pt#,
(0.264/0.580801)*pt#
```

Although it would be an odd design aspect, it is possible to standardize on a uniform depth, with the corresponding dimension adjustments as before. Finally, other standardizations are possible, with different corresponding numerical calculations on the dimensions, which would not change the overall look and feel of the font, as long as the aspect ratio $(height + depth/width)$ remains the same.

For our purposes here, let's use a constant width equal to the design size, and make all other dimensions proportional to the width. Even though the width is the same for each character in the font design, each glyph need not occupy the same

horizontal space available to a character.[1] The implicit and now relative form of lines 1–38 can now be seen.

```
100. path pth[];
101. designsize := 10;
102. ds          := designsize;
103. beginchar(99,ds*pt#,
104.           ds*(0.580801/0.4092)*pt#,
105.          (0.264/0.4092)*pt#);"myst";
106. x1=x2=0.594/(0.4092*ds)*w;
107. x3=x4+0.297/(0.4092*ds)*w
108.   =2.31/(0.4092*ds)*w;
109. x9+0.297/(0.4092*ds)*w
110.   =x10=x11+1.023/(0.4092*ds)*w
111.   =3.696/(0.4092*ds)*w;
112. x12+0.231/(0.4092*ds)*w=x11;
113. x12-x13=1.848/(0.4092*ds)*w;
114.
115. y1+(0.264/0.264)*d=0;
116. y2-0.957/(0.580801*ds)*h=0;
117. y3=y4-1.485/(0.580801*ds)*h
118.   =3.366/(0.580801*ds)*h;
119. y10=y11=y9-2.508/(0.580801*ds)*h
120.    =y12-1.782/(0.580801*ds)*h=0;
121. y12-y13=2.046/(0.580801*ds)*h;
122.
123.   [...]
124.
125. pth[1]=z1--z2;
126. pth[2]=z2..
127.        controls(1.716/(0.4092*ds)*w,
128.                 2.046/(0.580801*ds)*h)
129.             and(2.211/(0.4092*ds)*w,
130.                 2.805/(0.580801*ds)*h)
131.        ..z3;
132.
133.   [...]
134.
135. fill pth[1]
136.      for i=2upto6: & pth[i] endfor
137.      --cycle;
138. cull currentpicture dropping(0,0);
139. endchar;
```

Note that the `u` unit is no longer needed, as it is now a constant function of the chosen design size, and all dimensions are expressed in terms of the width, height, and depth, which are in turn multiples of the design size. Once again, the glyph produced by lines 100–139 looks exactly like that produced by lines 1–38 (so once again, see Figure 1). The fractions in the critical and control points need not be left as explicit reminders whence they came; simplified values, such as those found in the following excerpt, are often more convenient.

```
140. path pth[];
141. designsize := 10;
142. ds          := designsize;
143. beginchar(248,ds*pt#,
144.           ds*(1.419357)*pt#,
145.               0.645*pt#);"myst";
146. x1=x2=0.145161w;
147. x3=x4+0.07258w=0.564516w;
148. x9+0.07258w=x10=x11+0.25w
149.   =0.903226w;
150. x12+0.056452w=x11;
151. x12-x13=0.451613w;
152.
153.   [...]
154.
155. pth[6]=z12..
156.        controls(0.459677w,0.164772h)
157.             and(0.298387w,0.034091h)
158.        ..z13;
159.
160. fill pth[1]
161.      for i=2upto6: & pth[i] endfor
162.      --cycle;
163. cull currentpicture dropping(0,0);
164. endchar;
```

Analytical. The final transformation finishes the numerical calculations begun in earlier work. In particular, the explicit control points must be changed to implicit tension statements. This ensures that the glyph shape is defined by analytical considerations between critical points, and not by arbitrarily chosen "magic" control points.

The transformation from control points to tension statements is accomplished by the Matlab-based `cp2ab` utility.[2] Documentation for using this utility is included with its distribution. For example, for `pth[6]` (lines 155–158), using

```
z12,
(0.459677,0.164772*1.419357),
(0.298387,0.034091*1.419357), and
```

[1] This consideration is the basis for the `adjust_fit` function in plain METAFONT.

[2] If a user does not have access to Matlab, the freely available source code for `cp2ab` may be easily ported to many analytical calculation applications, such as `MAPLE` and `S`. A user may even use the code to make manual calculations.

```
z13
```

as input (so that all calculations take place in units of `ds*pt`), the corresponding tension statement would be

```
z12{dir-124.2156}
   ..tension0.9276and1.1893
   ..{dir-143.6158}z13
```

The `cp2ab` utility has the feature that all results are invariant under scaling. The `cp2ab` utility also accepts vector input (with corresponding vector output) so that all sequential post- and pre-tension values may be calculated simultaneously, as shown in this partial Matlab session.

```
z1 =
   0.1452 + 0.2339i
   0.5645 + 0.8226i
u1 =
   0.4194 + 0.5000i
   0.6129 + 0.9597i
[...]

[a1,b1,ppath1,diru1,dirv1]
       =cp2ab(z1,u1,v1,w1)

a1 =
    0.8004    0.7771
    1.0488   -1.0000
b1 =
    1.2963    1.3193
    1.2967   -1.0000
ppath1 =
   0.1452 + 0.2339i
   0.4486 + 0.5766i
   0.5645 + 0.8226i

   0.5645 + 0.8226i
   0.4919 + 1.1855i
  -1.0000
diru1 =
   44.1449   55.0882
   70.5599   -1.0000

[...]
```

The following excerpt shows the results of the analytical transformation, which still has not changed the appearance of the glyph from Figure 1.

```
pth[1]=z1--z2;
pth[2]=z2{dir44.1449}
..tension0.8004and1.2963
..{dir55.0882}
(0.4486*ds*pt,0.5766*ds*pt)
..tension0.7771and1.3193
..{dir79.9923}z3;

  [...]
```

Final product. To complete the METAFONT approach to font design, an effort should be made to minimize the number of tension values used in a glyph definition. This ensures a consistent look and feel from one character to another, especially along edges. For example, the use of 1.2963 and 1.2967 may be replaced by a single 1.3 value, such as in `pth[2]` and `pth[3]`. The final form of the mystery glyph, completely utilizing the METAFONT approach, may be found in Figure 5, and is listed in lines 211–258.

```
path pth[];
designsize := 10;
ds         := designsize;
beginchar(248,ds*pt#,
          ds*(1.419357)*pt#,
              0.645*pt#);"myst";
x1=x2=0.145161w;
x3=x4+0.07258w=0.564516w;
x9+0.07258w=x10=x11+0.25w
  =0.903226w;
x12+0.056452w=x11;
x12-x13=0.451613w;

y1+d=0;
y2-0.164772h=0;
y3=y4-0.255681h=0.579545h;
y10=y11=y9-0.431817h
   =y12-0.306818h=0;
y12-y13=0.352272h;

z5=(0.3951613w,0.846589h);
z6-z5=(-0.225806w,0);
z7-z6=(0,0.153409h);
z8-z7=(0.290323w,0);

pth[1]=z1--z2;
pth[2]=z2{dir44}..tension0.8and1.3
   ..{dir55}(0.4486*ds*pt,0.5766*ds*pt)
   ..tension0.75and1.3..{dir80}z3;
pth[3]=z3{dir70}..tension1and1.3
   ..{dir157}z4--z5--z6--z7--
   z8{right}..tension1and1
   ..{dir-34}(0.6652*ds*pt,1.3680*ds*pt)
   ..tension1.3and0.75
   ..{dir-76}(0.8026*ds*pt,1.1374*ds*pt)
   ..tension1.3and0.75..{dir-93}z9;
pth[4]=z9{dir-84}..tension0.75and1.3
```

```
248.    ..{dir-79}z10--z11;
249. pth[5]=z11{dir98}..tension0.8and1.3
250.    ..{dir94}z12;
251. pth[6]=z12{dir-124}..tension1and1.3
252.    ..{dir-144}z13;
253.
254. fill pth[1]
255.      for i=2upto6: & pth[i] endfor
256.      --cycle;
257. cull currentpicture dropping(0,0);
258. endchar;
```

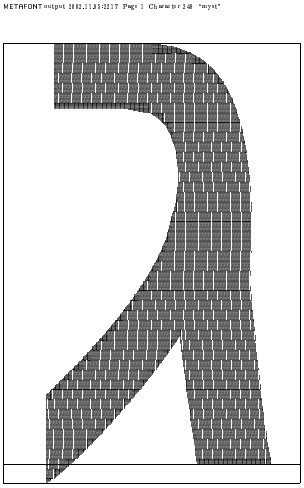


Figure 5: Final form

Summary. The METAFONT approach combines the irreplaceable advantages of implicit definitions (making the position of critical points refer to each other), relative policies (stating dimensions relative to the width and height of the bounding box), and analytical considerations (using tension statements rather than control points) to ensure the consistency and style of the resulting glyphs across all characters in a font.

With *practice, practice, practice,* and the right analytical tools, such font design far outshines, in flexibility and functionality, the let-us-do-it-for-you approach imposed by popular fontmaking applications.

◇ Timothy Hall
PQI Consulting
P. O. Box 425616
Cambridge, MA 02142-0012
info@pqic.com
http://www.pqic.com/TUG

Resources

CTAN plans

Robin Fairbairns, Jim Hefferon,
Rainer Schöpf, Joachim Schrod,
Graham Williams, Reinhard Zierke
ctan@dante.de

Abstract

The readers of *TUGboat* likely know the Comprehensive TEX Archive Network as a great pile of TEX stuff. That is, it is full of TEX materials and it is great, but it is also perhaps a pile — a bit of a mess.

We will sketch some plans for improving CTAN. As part of that, we will outline its architecture, history, and some issues.

1 Preamble

Taking it from the top: CTAN is an Internet archive of material related to TEX that is available for public download. We now hold five gigabytes of material. Each day about ten thousand visitors download a large number of files, and others upload some more.

We try hard to be definitive, to live up to our "Comprehensive" name. We hold everything from LATEX macro packages up to entire distributions such as MikTEX and teTEX.

2 Present

CTAN is not a single site, but instead is a set of sites. Three of these actively manage the material, for instance installing new or updated packages.

- dante.ctan.org in Germany is sponsored by the German TEX group Dante, and is maintained by Rainer Schöpf and Rienhard Zierke.
- cam.ctan.org is sponsored by UK-TUG and is maintained by Robin Fairbairns, in England.
- tug.ctan.org in the USA is sponsored by TUG, and maintained by Jim Hefferon.

To ensure that we have the same policies, we rely on an active mailing list. To ensure that we hold the same material, we rely on a number of custom scripts.

In addition to the core sites, about seventy-five sites around the world help out by offering a mirror — every day they sync up with a core site and then make their copy also publicly available. This gives users more options and eases network traffic. We encourage people to use the mirrors.[1]

[1] See http://www.dante.de/mirmon/ and also ftp://tug.ctan.org/tex-archive/README.mirrors.

3 Past

Before CTAN there were a number of sites with TEX materials available for download but there was no authoritative collection. At a podium discussion that Joachim Schrod organized at the 1991 EuroTEX conference, the idea arose to bring together the separate collections. (Joachim was involved because he ran one of the largest FTP servers in Germany at this time, and had heavily modified the basic tool `mirror.pl` for this purpose.)

CTAN was built in 1992, by Rainer Schöpf and Joachim Schrod in Germany, Sebastian Rahtz in the UK, and George Greenwade in the US (George came up with the name). The site structure was put together at the start of 1992 — Sebastian did the main work — and synchronized at the start of 1993. TUG provided a framework, a Technical Working Group, for this task's organization. CTAN was officially announced at the EuroTEX conference in Aston, 1993.

When CTAN was founded, the main way to access files over the network was FTP. So the system was built with an expectation that visitors would get materials that way (and perhaps also with an expectation that visitors are experienced users). In 1999 a try at a more extensive web interface was put on the TUG site, but it is weak and was not adopted by the other two core sites.

4 Problems

Nobody likes complainers, but to describe our plans we must describe the issues that they address. There are problems with the collection itself, and problems with the administration of that collection.

One problem with the collection is that it is big. Its structure has been outgrown and needs updating. Most people in the TEX community have had the experience of being unable to find the solution to a problem, only to later discover that a solution was in fact on CTAN. That is, we have found that as we have grown, the information available to archive users to help locate materials has not grown fast enough to allow them to find what they need. This has been eased by the metadata[2] assembled by Graham Williams into his *Catalogue*.[3] But nonetheless, we need to be more information-rich.

A longstanding request about the collection[4] has been for CTAN to keep histories of packages, so that users can compile documents that rely on old versions. (Now, when a package author sends an update, we overwrite the old material.)

Another problem in the didn't-think-we'd-get-big (or-old) category involves mirrors. Often, the best way for a user to get a package from CTAN is to get its entire directory at once, so that they don't miss some files. To that end, the core sites support on-the-fly creation of `.zip` and `.tar.gz` file bundles.[5] The web front end at `http://www.ctan.org` uses this capability, and something like it must be a part of any future interface. However, it doesn't work with mirrors. In order to send users who want a bundle to a mirror, the system needs to know which mirrors correctly do on-the-fly-ing. So we wrote a script to check. When we ran it we found that not one mirror actually made both `.zip` and `.tar.gz` bundles without error. Consequently, the great majority of downloads come from the three core sites.

The flip side of people getting things from us is our getting things from the community. We are concerned by a trend whereby some package authors do not upload their work, but instead leave it on a personal web server. This is bad because it brings us back to the pre-CTAN days of materials that are scattered and that may disappear; that is, CTAN could end up not-comprehensive. It is also bad because, even if we know that the author's work exists, we have trouble gathering this material since the web protocol HTTP makes it hard for us to fetch things into our holdings.[6] Obviously we must work with the world as it is, but this is a problem.

The final collection issue that we will mention is that early developers, including Knuth, expected that most users would be fairly sophisticated: they would have developed basic typographic knowledge, and would need a minimum of computer and development support (e.g., they would write their own macros). This has not proved to be so. We perceive instead that the majority of TEX users want a distribution that comes with LATEX, etc., already set up. If they need to get something else, then they would like the distribution to have a module that can interface with CTAN and set it up for them. So a key goal is that we must — in conjunction with distributions — produce a system that meets those expectations.

We next describe some issues with the administration of the archive. Users do not see these directly, but they have an effect on what users do see.

[2] Data about data, that is, information about the packages.

[3] `http://www.ctan.org/tex-archive/info/Catalogue`

[4] Notably by Nelson Beebe.

[5] For example, visiting the url `ftp://ftp.ctan.org/tex-archive/macros/latex/contrib/shadethm.zip` will get the entire `shadethm` directory as a zip archive. From a command line FTP client, `get /tex-archive/macros/latex/contrib/shadethm.zip` will do the same.

[6] On the scale at which we work, this is not as simple as just using a program like `wget`.

The first is that we are a shoestring operation. The machines have been granted by user groups, but the critical network connections are donated by each maintainers's institution. The maintainers are far apart — some mailing list members have never met any other member — which slows progress and adds chances for miscommunication. All of the maintainers are volunteers and satisfy CTAN's time demands in the face of other things that must come first.

The time demands on maintainers are relevant because they have slowed our development. In particular, we must promptly handle materials that are uploaded. To help a reader get a sense of it — and for the satisfaction of bellyaching — consider a new package upload. The machine's maintainer gets it from the upload area and unpacks it. He checks the license, and decides where the package will go. He checks that there is a `README` file, and that there is documentation in PDF format that uses Type 1 fonts. Often, these checks involve corresponding with the author or with the CTAN mailing list, introducing a delay of a day or more. He then uses our custom install script to copy the material into the public area, and to trigger the mirroring process. He notifies the CTAN announcement list (and thus `comp.text.tex`). Finally, he edits the package *Catalogue* metadata and puts it into the CVS tree. In all, it averages perhaps a half hour per package.

More people in the administration might help. However, in addition to the necessary expertise with TEX, systems administration, and with the layout of CTAN, our work takes place in the context of an increasingly complex computing world. To name one example, in recent years licenses have become a big issue. We need people, but we also need a way to bring them in so that they can learn gradually.

5 Plans

Suppose that a colleague gives you a paper that requires a package not in your TEX setup. At present, you would visit CTAN, find the package, and then install it. Imagine if, instead, your TEX distribution got the package, installed it, and proceeded with running the paper, without your having to know anything about it. Technology that would make this kind of negotiation between the user's computer and CTAN reasonable is called "web services".[7] Our most important goal is to develop — in coordination with existing distributions — a capable spectrum of web services for CTAN to offer.

One step toward that, and toward accomplishing present goals also, is to better organize our holdings. For instance, we have already combined the subdirectories `supported` and `other` of `/macros/latex/contrib`, and we plan also to meld the `/info` and `/help` directories. A bigger job is to break all of our holdings into packages, and have each package in its own directory (no more `misc`). This is the natural way to answer a web services query, "what is the latest version of the file `f` in the package `p`?"

In support of web services, and also to help visitors get more information out we must get more information into the *Catalogue*. We must both (1) expand the information of the kind that is in there already, and (2) also expand the kinds of information that can go in there.

Part of (1) is an effort to provide an easy way to edit this metadata on the web, for instance, when an author uploads or updates a package. As a bonus, this may provide a way to bring people in to help CTAN. A person could make a reasonable contribution by editing metadata and checking it into the CVS tree, without having to do system administration of a CTAN site.

An example of (2) is that we need to retain keywords, so that users can search for a package in this way (this search could happen on a CTAN web page, or from a user's desktop through a web service). For some time we've been discussing the underlying model for the metadata and in support of this, the *Catalogue* recently moved to a CVS tree.[8]

All that information should be in a database. This fits into our plans in many ways because, while CTAN grew up as an FTP archive, the web has changed everything and we need to fix our web system to be database-backed. It should provide an interface that is uniform across all three core sites.

That interface could allow users to find packages in alternate ways (this was first suggested by a comment made by Sebastian). At present, users can look through the FTP directories, can search the list of all files, can do a crude text search of the *Catalogue*, or can do a web search of our holdings using a standard search engine, and we've mentioned above that we'd like to add a keyword search. But, we'd also like to add a search of packages by functionality: a user trying to work with page headers might click through a branch of choices like `Top > LaTeX > Page layout > Headers and footers`.

One of the things that a modern site should have is the ability to search documentation. At present, many packages do not have documentation,

[7] A rough definition is: a web server will respond to queries beyond just requests for pages.

[8] `http://texcatalogue.sarovar.org`

or have it in a format that is not suitable as a search result (e.g., if the result of a search is a link to a `.dtx` file then clicking on it is unlikely to be helpful). We have begun enforcing that package contributers provide documentation just in PDF format, which is the only format that combines widespread accessability and typographic excellence.

Two problems listed above are the question of keeping package histories, and the question of mirrors providing `.tar.gz` and `.zip` bundles. We believe that we can solve these together, saving each version of a package as a bundle — then we have a bundle available, and mirrors need not create them.

We need to convince authors to upload their materials. We have in the past urged authors to do so,[9] but here also a volunteer, who can find materials and politely persuade authors, would help.

Finally, we are constantly thinking about the maintainer's work flow. There has been some wild talk about an administration GUI, but the problem is that there are so many exceptions and special cases that we often cannot see how to do it any other way than by hand.

6 Prognosis

We plan to make CTAN more of a "Comprehensible" TEX Archive Network. These plans have been under discussion and in development for two to three years.

Dante has helped out greatly by sponsoring key people to come to meetings in the last two years, at Bremen and at Darmstadt, for in-person discussions. We must say that even beyond the grace of the invitations, the Dante people were kindness itself: in particular, Volker and Klaus moved the entire process forward greatly.

At present, the *Catalogue* format has been adjusted to allow development (in addition, moving it to the CVS tree allows contributions in parallel), structures for the databases are in place, and we have beta code for the web editing of metadata and other parts of the new web system. Now, we must test that system. Also, we must supply the static data: the web page content, the keywords, the by-function categories, etc.. Finally, we need more data about packages for the database.

Briefly: progress is maddeningly slow, but there *is* progress.

[9] If you have something that others would find useful, please consider sharing it!

LaTeX

Some notes on templates

Lars Hellström

Over the last few years, the LaTeX3 project team has been making and releasing some packages belonging to what they call LaTeX $2_\varepsilon*$, which is a sort of intermediate step before LaTeX3. Unlike the previously released l3 suite of packages [2], which deals mainly with very basic programming structures such as lists and stacks, the LaTeX $2_\varepsilon*$ suite of packages is more about producing better results in more concrete areas of LaTeX. Examples include the xor package, which is a new and much more versatile output routine, and the xparse package, with which one can easily define commands with complicated mixtures of mandatory, optional, and *-type arguments. The elegant interfaces and functionality of these packages promise well for the future.

Yet, most of them are still in a rather experimental state and they are currently only available from the experimental code directory on the LaTeX project web site [4]. Some of the packages are however not too far away from a more general release — in the event of which they will probably become part of the Required suite of LaTeX packages — and the first of these will most likely be the template package [1]. This is a very interesting package, because it provides the means for a whole new (and very promising) style of LaTeX programming, which I had the opportunity to try out during my work on the docindex package [3]. This note is an attempt to summarize the observations I've made about this programming style, in the hope that it may guide others in their first experiences of it. I am quite convinced it will become an important part of LaTeX3 and also of the further development of LaTeX 2_ε.

Even though the name of it all is template, one shouldn't overrate the importance of the templates. Most of the things you actually keep around are not templates, but *instances*, and an instance is basically a very familiar object: a macro which performs some action. Of course, in TEX programming almost everything is a macro in one way or another, but some things are macros only because TEX doesn't provide any better way of storing some kinds of data, and some macros only exist to help other macros parse their arguments. Instances are rather the kind of macro you write because you want to modularize your code; typical actions are to typeset a float caption and to set the paragraph justification.

An instance is not just any nice collection of simpler commands, though. To begin with, every instance has a *type*, which specifies the syntax and most of the semantics of the instance. All instances with the same type do roughly the same thing, but they usually differ in the details. If one instance of a certain type, say, takes some text and typesets it as a section heading, then another instance of the same type could typeset the same text as a subsection heading or a part heading. If one instance of a certain type causes the index to be typeset (in the same sense that `\printindex` does) then another instance of that type might cause the glossary to be typeset. The simpler an instance is the more detailed the type specification usually gets, but it is generally about *what* the instance does, not about *how* it does it. Two instances of the same type are exchangeable in the sense that replacing one with the other doesn't cause any errors, although it will almost certainly change the typeset appearance of something.

That the instances are typed is incredibly useful, because it means you can redefine any instance without having to worry about breaking anything (as long as your redefinition conforms to the type specification)! An average LaTeX package usually comprises a couple of user level commands, a couple of parameters, and a number of private macros. The user level commands and parameters tend to have well-defined syntax and semantics, even though the choice of parameters is often less than satisfactory, but the syntax and semantics of the private macros are generally something about which everyone but the package author knows very little. There is often no other way to achieve a certain layout modification than to redefine a private macro, but that is always a risky operation because inferring the semantics of something from its implementation is no exact science.

In an implementation employing template techniques, many of the private macros would instead be instances, and thus this wouldn't be a problem; the type specification would say everything one needs to know. In the case of new types there is of course a bit of extra work needed for writing down the specification, but that work is usually well spent as it helps pointing out weaknesses in the implementation. An interesting side-effect of using instances is that there is much less need for package parameters, as many of these can instead be embedded into the instances which are easily redefined. Thus the net effect of using template techniques in a package can actually be that the interface to the package becomes simpler as well as more powerful and versatile.

The other thing about instances is how they are defined; this is where the templates get into the picture. A *template* is also basically a macro, and like an instance it has a type, but a template additionally has an associated set of parameters. Instances are defined by specifying a template (of the correct type) and the values that the parameters of the template should have. A typical instance definition looks something like

```
\DeclareInstance{justification}
  {flushleft}{std}{
    rightskip  =0pt plus 1fill,
    leftskip   =0pt,
    startskip  =0pt,
    parfillskip=0pt plus 1fill
}
```

Here `justification` is the type, `flushleft` is the name of the instance being defined, and `std` is the name of the template it is based on. The last argument is a keyval list of parameter names and values. What happens internally is that a couple of control sequences (the respective *storage bins* for the parameters of the template) are set to these values whenever the instance is used and the code in the template accesses the values by using the storage bins in very much the same way as we currently use a package parameter.

A parameter value need not be a length or some other numeric quantity, however; it can just as well be a function (which in this case is a fancy name for a command), a name, a boolean, or another instance (usually, but not necessarily, of a different type). A very common use for function valued parameters is to handle formatting of short pieces of text; for example, the heading of the `theorem` environment. A template which handles this might for example have a function valued parameter `heading-format` which receives the theorem number as its only argument. Then to get headings in the default "**Theorem 6.2**" style one could say

```
heading-format = \textbf{Theorem~#1}
```

whereas the reverse "**6.2 Theorem**" style would be the result of

```
heading-format =
    \textbf{#1\hspace*{0.5em}Theorem}
```

Generalizing slightly, one could also imagine there being a function parameter `named-heading-format` with two arguments which is used instead of the `heading-format` parameter when the theorem has a name (e.g., "Inverse Function Theorem"). Passing

this name as the second argument, a suitable value for `named-heading-format` in the first case might be

```
named-heading-format =
    \textbf{Theorem~#1 (#2)}
```

or even

```
named-heading-format =
    \textbf{Theorem~#1 (\textit{#2})}
```

whereas the second value for `heading-format` might go better with

```
named-heading-format =
    \textbf{#1\hspace*{0.5em}#2}
```

e.g., "**6.3 Inverse Function Theorem**".

I personally find the separation of code into on one hand a template and on the other hand values of the parameters of that template quite a relief, because it physically separates two different levels of programming. The actual template usually only contains the "hard", programming-like parts of the code — arithmetic, decision-making, interpretation of arguments, and so on — whereas "soft" parts like the layout specification are put in the parameters. This means whenever the code actually does something that is directly visible in the layout — such as insert a skip or format a heading — it simply uses the value in a parameter or passes the relevant data on to a parameter for further processing. The resulting code in the template looks very much like a skeleton of only the hard parts with some sprinkled markers saying "insert soft thing doing ... here", but it is actually complete and working. It cannot be used until the parameter values have been specified as well, though, and the normal way of doing this is to define an instance of the template.

Since the soft programming of selecting values for parameters is much more like writing a LaTeX document than the hard programming, it is not surprising that the many LaTeX users who have not mastered the hard programming will find that their ability to modify the behaviour of a package is much higher for templated packages than for traditional ones. For those who have mastered hard programming the advantages may seem less clear, but my personal experience was that programming became simpler. How can this be if I am actually restricting the ways in which I may write the code? I think it has to do with how one thinks about the problem at hand. When one is doing hard programming one also gets into a "hard" mode of thinking, whereas when one is doing soft programming one gets into a "soft" mode of thinking. In the traditional style the hard and soft parts are often heavily mixed and consequently one is forced to constantly switch between two modes of thinking. In the templated style the mixing is much less pronounced due to the aforementioned separation and consequently one does not have to switch mode that often. As it does require some effort for the mind to switch mode, the less one has to switch the better!

There are many other things which could be said about the template package — how one can use calc type expressions as parameter values, how one can use collections to effectively have several definitions of an instance in memory simultaneously and quickly switch between them, what one actually does to declare a new template — but these are things one can easily find in the template package manual. I certainly hope that you will give it a try, because this is one of these things after which LaTeX programming will never again be quite the same.

References

[1] David Carlisle and Frank Mittelbach. The template package. Available from `http://www.latex-project.org/code/experimental/template.tgz`, 1999.

[2] David Carlisle, Chris Rowley, and Frank Mittelbach. The LaTeX3 programming language — a proposed system for TeX macro programming. Available from CTAN, `macros/latex/exptl/project/expl3/`, 1998.

[3] Lars Hellström. The docindex package. Available from CTAN, `macros/latex/contrib/xdoc/docindex.dtx`, 2001.

[4] Various authors. LaTeX project web site directory for experimental code. Located at `http://www.latex-project.org/code/experimental/`, 1999–present.

◇ Lars Hellström
Sand 216
S-881 91 Sollefteå
SWEDEN
`Lars.Hellstrom@math.umu.se`

Writing a big book — A first experience with LaTeX

David Walden

This paper is in response to TUGboat editor Barbara Beeton's Editorial Wish List on the TUG web site for papers by non-experts for non-experts. I also thought my experience might be interesting because many other people have had to struggle with a book or journal publisher that has no interest in LaTeX; while my approach is quite ad hoc with many manual rather than computerized steps, such an approach can be (or at least seem to be) faster than seeking a fully computerized approach.

Section 1 describes why I decided to write a substantial book in LaTeX, absent prior experience with LaTeX, starting from a manuscript of a prior edition of the book prepared in Word, for a publisher with no knowledge in LaTeX. My steps included converting the prior Word manuscript to LaTeX (section 2), massively changing the manuscript in LaTeX to produce a fully designed manuscript output in PDF format (section 3), and then converting the LaTeX via HTML into Word files containing ASCII text for the publisher to input into QuarkXPress for production of the pages to be sent to the printer (section 4). I conclude with some reflections on my process (section 5).

1 Background

Several years ago, the publisher of a text book [1] I had co-authored asked for a second edition. The original book was 575 pages long. Because of advances in the field since the first edition's publication, the second edition would be significantly longer and quite different from the first edition. In effect, we would be writing a new 755 page book [2].[1]

At about the same time the request came from my publisher, my frustration with composing large documents in Microsoft Word was reaching a peak. Because of new releases of Word and switches I made from using Word on early IBM PCs to using it on Macs to using it again on circa 2000 PCs running Microsoft Windows, I was having trouble accessing Word files I had created years before. I vowed never again to create a large document that had proprietary[2] formatting information as Word has; I wanted ASCII text with explicit markup commands, so I could reprocess a document at any later time using a text editor and its macro capability or by writing a small text transformation program (for example, in Perl).

Also, I didn't know how to make Word do the following sorts of things (if it can do them at all):

- automatically adjust chapter, section, figure, etc., numbering as chapters, sections, figures, etc., are added, subtracted, and reordered;
- automatically order table of contents entries and insert page numbers;
- automatically adjust cross-references to chapters, sections, figures, etc., as necessary;
- support a file of include-file commands to permit working with various portions of the book at different times;
- allow global switching on and off of various manuscript components (such as marginal notes to myself about what still needed to be done);
- including EPS files of figures at specified points in the manuscript while avoiding awkward page breaks.

Therefore, I decided to learn and use LaTeX which I had long known about but never used.[3] My learning of LaTeX was a bit hit and miss, as I started rewriting my book.[4] Over time, I did the following:

- bought some books on TeX and LaTeX;
- found `www.tug.org` and thus the MiKTeX collection of software (see `www.miktex.org`);
- began watching the `comp.text.tex` newsgroup and thus stumbled across the series of lectures on LaTeX and PDF by David Arnold of California's College of the Redwoods;[5]
- learned of WinEdt (`www.winedt.com`);
- configured WinEdt to run LaTeX, run DVIPS, and Distill the result into PDF files.

2 Moving the first edition book into LaTeX

I got the first edition Word files back from the publisher and repeatedly used WinEdt's Find and Replace commands to convert the Word file for each chapter into a LaTeX file:

[1] Throughout all of the steps described in what follows, my co-author Shoji Shiba worked collaboratively with me to create the content of the book; however, I did all the keyboarding and other computer-based work.

[2] Invisible and not able to be made visible, undocumented or unpublished, totally controlled by Microsoft.

[3] I had used programs such as RUNOFF (under the CTSS operating system), MRUNOFF (under TENEX), and TROFF-NROFF (under UNIX).

[4] There were many more steps in learning LaTeX and rewriting my book than I describe in this paper, including plenty of missteps. In this paper, I skip much confusing detail and simplify the ordering of steps a little to allow the reader to understand the main thread of my story. Any reader who wants more detail should ask me for it.

[5] `http://online.redwoods.cc.ca.us/instruct/darnold/index.htm`.

- open and close smart double quotes from Word were changed to `` and '', and open and close smart single quotes were changed to ` and ';
- em-dashes were changed to ---;
- hyphens between numbers were changed to --;
- footnotes (with the invisible, proprietary Word markup) were changed to use the `\footnote` command in LaTeX;
- the escape character \ was inserted before $, %, etc.;
- ellipses (three periods in a row) were changed to the `\dots` command;
- chapter and section titles were enclosed in appropriate `\chapter` and `\section` commands;
- the command
  `\addcontentsline{toc}{chapter}{`*title*`}`
  was used to add table of contents lines without chapter numbers for the *title* of the five major parts of the book;
- italic text was enclosed in `\emph` commands, bold face text was enclosed in `\textbf` commands, various non-English letters were handled with a \ and appropriate diacritical mark, etc.

The steps just listed took half an hour or so per file for each of the 20 or so files of the first edition book.

Next, I wrote some new LaTeX environments and commands, including the following:

- `blist`, `nlist`, and `dlist` environments for enumerated, numbered, and description lists with tighter inter-item spacing than the LaTeX default;[6]
- `\mnote{`*text*`}` command to put *text* to myself in the margin and `\CK` command to put a bold **CK** in the manuscript to highlight facts that needed checking;
- `\snfig`, `\snufig`, `\swsnfig`, etc., commands to include EPS figures in the text with the option of having them numbered and captioned, unnumbered and uncaptioned, sideways on a page, etc., and a similar set of commands to insert tables[7] in the text. See Figure 1 for an example of the definition of one of these commands.[8]

```
\newcommand{\snfig}[3]{ %scaled numbered figure
  \begin{figure}[htbp]
    \hfil\scalebox{#3}{%
    \includegraphics{figures/fig#2.eps}%
    }\hfil
    \caption{\label{fig:#2}#1%
      \texttt{\small[fig#2]}%
    }
  \end{figure}
}
```

Figure 1: Definition of `\snfig`

Once these environments and commands were available,

- I converted the lists in the original manuscript to use the appropriate new environments by typing the necessary `\begin`, `\end`, and `\item` statements into each file of the manuscript.
- I converted absolute figure and table references from the first edition into references to labels, using `\ref` and `\pageref`.
- I converted absolute chapter and section references to `\ref` and `\pageref` references to appropriate labels I had inserted in the LaTeX versions of the chapter files.[9]

Finally, I compiled the LaTeX files and printed them out to compare them with the first edition book, looking for instances where I had missed something that needed to be converted to LaTeX.

3 Rewriting and updating the book

With the first edition fully converted to LaTeX as described in the previous section, it was time to move on to updating the book.

I drew additional text from other papers I and my co-author had written since the first edition of the book was published, doing the same brute force but fast enough conversions from Word to LaTeX. We wrote lots of new text that I typed into the LaTeX

[6] After I finished the effort described here, I learned about the `memoir` class that includes support for tighter spacing of list items.

[7] I used LaTeX's `\tabular`, etc., features to format a few simple tables. Mostly, however, I already had tables or created tables as EPS files with Illustrator.

[8] The macro for `\snfig` inserts the figure using the `graphics` package with parameter `dvips`, creates a label for it derived from the figure's file name for cross-referencing from elsewhere in the book, and prints enough of the figure's file name in square brackets on the page to which the figure floats so that I could find the figure's file when I saw a figure needing a change.

The macro as shown here has extra new lines (so it fits within a single column) that were not in the macro as I actually used it.

[9] I didn't redefine the `\chapter` and `\section` commands to automatically produce appropriate labels because I didn't want to take the time to learn how to do this and discover whether it was actually a good idea or not. Perhaps I should have. Generally I was focused on getting the new book written and sent to the publisher rather than on learning to be a LaTeX expert.

files. We created many new figures and tables that I input into Illustrator files.[10]

An important aspect of the book was a substantial number of real life case studies and examples. Thus, I decided to include a table of case studies (to go after the table of contents), listing the title of each case study and the page number on which it began. Rather than learn how to define a LaTeX construct that worked like a table of contents such that a new entry was automatically added each time a new case study was added, I simply typed the case study name and a `\pageref` into a case-study-list file in the order of the case studies in the book. When I rearranged parts of chapters so the order of case studies changed, I had to do a parallel rearrangement of the lines of the case-study-list file. It was easy to see when rearrangement of the case-study-list file was needed by the out-of-order page numbers.

Eventually, I was ready to show a first draft of the new book to the development editor from the publisher's staff. I used the command

```
\setlength{\parskip}{.24in}
```

to (sort of) produce a double-spaced copy of the manuscript for the development editor to review.[11]

Based on the development editor's comments, I further revised and reordered the manuscript. In particular, the editor asked that a single bibliography be created as part of the backmatter of the book to which cross-references could be made from the main text, a significant change from the first edition which included citations in the end notes for each chapter. Thus, I began to use BibTeX, creating a `.bib` file of all bibliographical items and replacing citations in the text with appropriate uses of the `\cite` command. If there is one thing I wish I'd done differently in my work on the new book, it is to use BibTeX and a `.bib` file from the start. Note only did it eliminate lots of chapter end notes; it also provided a single extensive bibliography, arranged alphabetically by author, which has been a valuable addition to the book.

In time,[12] a final manuscript was available. Again, I provided double-spaced pages for copy editing. However, by this time, the publication deadline was looming, so as I finished the final draft of each chapter, I sent it as an email attachment to the copy editor, who did his work and then Fedexed red-marked pages back to me from which I made appropriate changes in the LaTeX files.

4 Producing the final book

With a finished manuscript in hand, it was time to refine the book design so the publisher could see precisely what I wanted. This required using the `fancyheadings` package to set up page headers and footers, using the `titlesec` package and then making a tiny change to a private copy of the `titlesec` style to adjust chapter headings, and slightly redefining (in the LaTeX preamble) LaTeX's book class (`\@makecaption` and `\thechapter` as used in table and figure captions) to get captions to look as I wanted. I learned how to do these redefinitions by sending a query to `comp.text.tex`.

It was also time to discover the format the publisher wanted the final manuscript in — a topic I had been avoiding because I didn't want to hear any back pressure against my use of LaTeX for the book rewrite. I offered some choices to the publisher's production manager for the book:

1. I could finish the job in LaTeX (getting rid of strange page breaks, etc.) and deliver a ready-to-print PostScript file for the book.
2. I could convert the manuscript back to Word, fully formatted (I didn't know how I would accomplish this with other than *lots* of manual work).
3. I could do something in between 1 and 2.

The publisher's layout person and I communicated. He stated that he would be using QuarkXPress to lay out the book. Once he understood what I had in LaTeX, he asked me to provide him with ASCII text in Word files with no formatting: he wanted Word files because in his experience special characters such a non-English vowels including diacritical marks translate best into XPress from Word; he wanted no formatting so he wouldn't have to remove anything unnecessary before adding formatting in XPress.

My problem, therefore, was how to convert the *output* of LaTeX (with all the benefits of LaTeX's chapter, section, figure, citation, etc., numbering and cross-referencing) into Word files (without any LaTeX markup, LaTeX-produced hyphenation, etc.). I decided that moving from files with LaTeX markup through HTML produced by LaTeX to ASCII in Word

[10] The publisher did the first edition figure finish work in Illustrator from MacDraw Pro figures I submitted with the Word manuscript. Therefore, I used Illustrator to draw new figures and tables and to modify old figures and tables. Altogether, the finished book had hundreds of figures and tables in EPS format.

[11] I always did my own proofreading and reviews of what I was writing using a single-spaced copy of the manuscript because it helped me better picture the final book I was aiming toward.

[12] The total elapsed time to rewrite the book was about 18 months, most of which was involved with content rather than formatting.

files would result in what the publisher's layout person needed.

Therefore, I spent a few days (*just* a few days) doing the following:

1. Sent the layout person a single-spaced copy of the fully designed and fully formatted book out of LaTeX so he had that to guide him as he did his work in XPress.
2. Replaced the `\footnote` commands in all 30 or so LaTeX files with instances of a newly defined `\enote` command that produced end notes following each chapter, using the `endnotes` package.[13]
3. Modified my figure- and table-producing commands (like `\snfig` shown in Figure 1) so they dropped the marginal notes indicating the figure file names, did not actually insert the EPS file, and instead just included the figure number and caption on its own line at an appropriate place in the manuscript along with the file name of the figure or table. The locations of the modified figure- and table-producing commands were always immediately after the the paragraph of first reference to the figure or table. Thus, the person doing the layout had the information necessary to place each EPS figure or table with its number and caption at an appropriate place on an appropriate page.
4. Added the command
   `\setlength{\defaulthyphenchar}{-1}`
   in the preamble to turn off hyphenation to avoid artificial within-word breaks within paragraphs.
5. Bought a copy of VTeX, the advertising for which claimed it to be the best program for generating HTML from LaTeX because it generated HTML from the LaTeX itself and not from DVI output of LaTeX.[14]
6. Changed a few small items to conform with VTeX, e.g., from using the `graphics` package I had been using with MiKTeX to using the `graphicx` package.
7. Got a small modification to VTeX from MicroPress Inc. so that VTeX generated no extra characters in its HTML output (e.g., no gratuitous spaces surrounding bibliographic reference numbers).
8. Processed each LaTeX chapter individually (using `\includeonly`) through VTeX to produce an HTML file for the chapter.
9. For each chapter, selected and copied (using control-C under Windows) the entire text of the HTML output file and pasted it (using control-V) into a new WinEdt file.
10. Configured WinEdt to have essentially infinitely long lines of text and touched each paragraph with the mouse and keyboard so that each paragraph became one line. In other words, I removed intra-paragraph new lines that would interrupt the flow of text on to pages in XPress.
11. Selected and copied (again using Windows' control-C command) all the text for the chapter in WinEdt and pasted it into an empty file in Word (using control-V). Somehow moving all the text from the HTML file to WinEdt and then to Word discarded all of the HTML markup commands and left only ASCII text of the manuscript in the Word file, which wouldn't have happened with a direct copy from the HTML file to Word.
12. Reviewed the Word file for anything that might confuse the layout person and clarified it.[15]
13. Sent each Word file to the layout person to cut-and-paste into XPress and there to format it for printing.

The layout person did the layout of the whole book and then sent it to a proofreader. The proofreader reviewed the entire set of page proofs, marked them where corrections were apparently needed, and sent them to me. I reviewed the proofreader's notes, changed my LaTeX files appropriately (to keep them in sync with the book as published), and sent the page proofs back to the layout person with instructions to ignore the few changes from the proofreader I didn't want made. Somewhere around this same time, the page proofs went to the indexer who created the index and sent it to the layout person to include at the end of the book.[16]

[13] While I was writing the new book I used bottom-of-the-page footnotes because of their proximity to the text they augmented. However, my publisher preferred all such notes to be at the end of the book as a collection of end notes, to avoid having many footnotes on text pages that might make the book look quite technical and, thus, less popular. We compromised on having end-of-chapter end notes.

I made a couple of tiny modifications to a private copy of the `endnotes.sty` file in order to format the chapter end notes as I wanted them to be.

[14] I have heard that other software packages, such as `latex2html`, also generate HTML directly from LaTeX.

[15] Somewhere along the line, about this time I formatted the output of `\tabular` commands so the layout person would recognize it and format the content with XPress.

[16] I may misremember some of the ordering of the passing of nearly final manuscript from person to person. In any case, what I have described in this paragraph is what, in effect, happened (including no use of LaTeX's indexing capabilities).

5 Reflections

5.1 On using LaTeX

I surely could have been smarter about the way I used LaTeX. I feel like I could have made more use of redefining environments and commands. I could have done a better job of making it possible to change one environment or command definition to change the behavior (e.g., switching it off or on) of all instances of use of the environment of command. I learned to do almost nothing with fonts other than use the built-in ones and switch among their standard modes (and, since my book had no math in it, I learned nothing about LaTeX's math layout capabilities). I am sure there were "right ways" to do things that I did with the first string of commands or characters that I stumbled across that achieved the effect I wanted.

I will welcome suggestions from readers about smarter use of LaTeX.

5.2 On writing a book in LaTeX

The choice to rewrite the new edition of the book in LaTeX is one I would make again. Conceivably it would have taken less overall effort to do it all in Word.[17] However, I would not have been as happy as I was seeing and flexibly reorganizing and rewriting the book-like manuscript with embedded figures I had with LaTeX. (Also, I benefited by learning a lot about LaTeX.) The final conversion from ASCII text in LaTeX files to ASCII text in Word files for input to the layout person and XPress was a simple and quick enough step, given the VTeX capability to generate HTML without extra HTML formatting.

I will welcome critiques and suggestions for a better overall strategy for creating the design and manuscript of a big book in a way that provides me as author with lots of flexibility and a thorough image of the final product while at the same time working with a publisher apparently unacquainted with the likes of LaTeX and not used to authors so willing to get deeply into the computer aspects of creating the files to be sent to the printer.

[17] It might also have been more efficient for me to do the entire rewrite of the manuscript in XPress for continuity with the final layout work by the publisher. However, at the beginning of the project the publisher's art department was adament that an author could not be allowed near XPress — that was a domain reserved for the publisher's layout people.

Acknowledgments

The publisher's production editor, Michael Ryder, and composition person, William Brunson, cooperated completely in my effort to convert the LaTeX files into files the publisher needed to have the book printed.

Michael Vulis of MicroPress Inc. quickly provided a needed small change to VTeX. Peter Flynn answered my query to the `comp.text.tex` newsgroup regarding making captions look as I wanted them to. I also used many other resources available to the TeX community, for example, at CTAN and on the TeX Live CDs that come with membership in the TeX Users Group (along with a subscription to *TUGboat*).

Of course, many other people helped with the non-LaTeX-related aspects of writing the book. They are acknowledged in the book itself.

I appreciate the guidance for preparing this paper for publication that I received from Barbara Beeton, Karl Berry, and from the anonymous reviewer.

References

[1] Shoji Shiba, Alan Graham, and David Walden. *A New American TQM.* Productivity Press, Portland, OR, 1993.

[2] Shoji Shiba and David Walden. *Four Practical Revolutions in Management.* Productivity Press, Portland, OR, 2001.

Author's biography

Over a thirty-plus year career in high tech computer R&D, David Walden worked successively as a technical contributor, a technical manager, and a general manager. Most of this time, he was with Bolt Beranek and Newman Inc. of Cambridge, MA, where he was part of the original ARPANET development team and involved in many other early Internet activities. He has done much writing on technical, management, and other topics.

◇ David Walden
12 Linden Road
E. Sandwich, MA 02537
dave@walden-family.com
http://www.walden-family.com

Designing packages for Λ: An overview

Apostolos Syropoulos

1 Introduction

The Ω typesetting engine was introduced about ten years ago [2]. Roughly speaking, it is a Unicode TEX extension, and, in our opinion, it is the best TEX extension available. But even after ten years, there are certain things which remain undocumented. For example, there is no single document describing how one can prepare a Λ package! However, this lack of documentation is somehow justified since Ω is still an evolving system. Thus, sometimes it makes no sense to document an experimental feature that may not be present in the next release of the system. Still, there are certain features that are frozen and so we need proper documentation for at least these features.

Generally speaking, any LATEX package is a Λ package, but the inverse does not hold. The previous assertion is true just because Ω provides the so-called Ω Translation Processes (ΩTP, for short), which are used to transform the character encoding of the input stream and are among the novelties introduced in Ω. Thus, new Λ packages, whose functionality relies on ΩTPs, should provide the "familiar" user-interface (i.e., thru package options and commands) for their activation/deactivation.

It is our belief that Λ packages should accept Unicode encoded files just like TEX accepts ASCII files. In practice, this means that we need Unicode-encoded fonts. Since Type 1 fonts do not provide such a facility, and the use of OpenType fonts is still an item of active research, we need to create Ω virtual fonts in order to create virtual Unicode fonts.

In this paper, we begin by presenting a few particulars about the Inuit people and their language. Next, we present the general functionality of the oinuit package. We continue with the implementation details of the package. In particular, we describe how we implemented the various package options, the language switching commands, and the design of the various ΩTPs involved. In addition, we outline the implementation of Ω virtual property list files, which are used to create virtual fonts, and we finish with a description of the implementation of the hyphenation rules of the Inuktitut language.

2 Inuits and their language

The Inuit (here we also include the closely-related Yupik) are a native people of the Canadian Arctic, Greenland, Alaska, and the Chukotka Autonomous Okrug of the Russian Federation. Inuktitut (the language of the Inuit) and Yupik together form the Eskimo branch of the Eskimo-Aleut language family. The Eskimo branch is estimated to have 73000 speakers at present. Although linguists continue to use the term Eskimo, the people themselves prefer the term *inuit*, which is the plural form of the word *inuk*, meaning "human being".

Morphosyntactically, Inuktitut is an agglutinative or polysynthetic language. This means that multiple morphemes combine into what can be called words, which represent concepts that may require entire sentences in other languages. Inuktitut also features a morphological process called incorporation. A fuller discussion of these topics is beyond the scope of this paper.

James Evans, a Wesleyan (Methodist) missionary, is the creator of the Inuit syllabary. This writing system was initially created for the Ojibwe language, based on Pitman shorthand. Later Evans learned the Cree language and adapted his syllabary to write a translation of the New Testament in the Cree language. Rev. Edmund Peck adapted Evans syllabary and introduced it to the Inuit people at Little Whale River in 1876.

It is interesting to note that the syllabics are used by Inuit who live in Canada, especially in the new Canadian territory of Nunavut. On the other hand, Inuit in (what is now) the Northwest Territories, Labrador Coast and in Alaska use the Roman alphabet, as do the Inuit of Greenland (Greenlandic). Siberian Inuit use the Cyrillic script to write Inuktitut. Unfortunately, the use of the Inuktitut language has declined in those areas where syllabics are not used (with the lone exception of Greenland). In Table 1 the reader can view the Inuktitut syllabary currently in use as well as the Latin transcription of each symbol. For more information on the history of the Inuktitut syllabary the reader should consult the excellent article by Kenn Harper [1].

3 Typesetting Inuktitut with Λ

We now briefly describe the functionality of the oinuit package.

The package provides five options: `nunavut` (default option), `quebec`, `iscii`, `utf8`, and `ucs2`, which respectively correspond to source text using the Latin transcription of Inuktitut and the Anglican orthography, the Latin transcription of Inuktitut and the Catholic orthography, the Inuit ASCII (see Table 2), the UTF-8 Unicode encoding, and the UCS-2 Unicode encoding. Note that the Inuit ASCII is based on a PC Inuit Character Table proposed by Everson Typography (see the page at `www.evertype.com/standards/iu/iu-tables.html`).

ᐃ	i	ᐅ	u	ᐊ	a	ᕼ	h
ᐱ	pi	ᐳ	pu	ᐸ	pa	ᐦ	p
ᑎ	ti	ᑐ	tu	ᑕ	ta	ᑦ	t
ᑭ	ki	ᑯ	ku	ᑲ	ka	ᑲ	k
ᒋ	gi	ᒍ	gu	ᒐ	ga	ᒡ	g
ᒥ	mi	ᒪ	mu	ᒪ	ma	ᒻ	m
ᓂ	ni	ᓄ	nu	ᓇ	na	ᓐ	n
ᓕ	li	ᓗ	lu	ᓚ	la	ᓪ	l
ᓯ	si	ᓰ	su	ᔅ	sa	ᔅ	s
ᔨ	ji	ᔩ	ju	ᔭ	ja	ᔾ	j
ᕆ	ri	ᕈ	ru	ᕋ	ra	ᕿ	r
ᕙ	vi	ᕚ	vu	ᕝ	va	ᖁ	v
ᖃ	qi	ᖅ	qu	ᖃ	qa	ᖅ	q
ᖏᒋ	ngi	ᖏᒍ	ngu	ᖏᒐ	nga	ᖏ	ng
ᓬ	lhi	ᓭ	lhu	ᓫ	lha	ᓴ	lh
ᖖᒋ	nngi	ᖖᒍ	nngu	ᖖᒐ	nnga	ᖖ	nng

Table 1: The Inuit syllabary and its Latin transcription.

Also, note that when using the Anglican orthography, one places a dot over a symbol to denote that the vowel of that syllable is "long"; whereas when using the Catholic orthography, the difference in vocalic length is indicated by duplicating the symbol for the vowel which is long.

To assist people who happen to use an ASCII editor to prepare their documents, we defined a few commands that can be used to switch languages and fonts. The command `\textinuit` assumes that its argument is a piece of Inuktitut text that is typeset accordingly. Similarly, the command `\inuittext` changes the internal state of Λ and everything from now on is assumed to be Inuktitut text. The environment `inuit` does exactly what the command `\textinuit` does. In addition, it is possible to switch between languages with the `\selectlanguage` command. Notice that all these commands implement the functionality of the corresponding commands that the LaTeX babel package provides. Naturally, our aim was to provide an "established" interface and not to re-implement the babel package.

Last but not least, the command `\InuitToday` is the Inuktitut version of the `\today` command.

4 The implementation details

We believe that good software should always have good documentation. Apart from creating TeX and METAFONT, Donald E. Knuth created the so-called *literate programming* methodology for program development. Roughly, this methodology is based on the observation that when one describes what he wants his program to do, he can implement it more easily. This program methodology is an offspring of the structured programming methodology of the 1960's. Although nowadays there are many new program development methodologies (e.g., generative programming), we still believe literate programming is quite adequate for the development of LaTeX/Λ packages. So we decided to implement our package using the literate programming tools for LaTeX (i.e., the `doc` package and the `docstrip.tex` TeX program originally developed by Frank Mittelbach).

Another decision we had to make was which character set to use. Naturally, since the "default" character set for Ω is the UCS-2 character set, one may opt to use this set. However, UCS-2 encoded files cannot be viewed "out of the box" on most computer platforms. Practically, this means that one should stick to good old ASCII even when developing Λ packages. Of course, many readers will object to this idea, but for the moment I believe this is the best option for development of packages for Λ.

Now we proceed with the various implementation details. In what follows we assume familiarity with ΩTPs. Readers not familiar with ΩTPs should consult the Ω documentation (e.g., see [2]).

4.1 The macros

By default, Λ uses the UC font encoding for monospaced fonts. However, it is quite possible that the user has a Λ format built without the necessary patch, so we first need to make sure that UC is the default font encoding for monospaced fonts:

```
\IfFileExists{ot1uctt.fd}{%
   \def\ttdefault{uctt}}{}%
```

As we noted in the previous section, the oinuit package offers a number of options. Here we describe how we implemented this facility. We present the relevant code for only two options for reasons of brevity:

```
\DeclareOption{nunavut}{%
   \ocp\InInuit=Qinuit2uni
   \ocplist\InInuitList=
       \addbeforeocplist 1 \InInuit
      \nullocplist
}
%
  ..........................
%
\DeclareOption{ucs2}{%
   \ocp\InInuit=id
   \ocplist\InInuitList=
       \addbeforeocplist 1 \InInuit
      \nullocplist
}
%
```

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
20	SPC	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
60	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL
80	Δ	Δ̇	ᐅ	ᐅ̇	ᐊ	...	ᐊ̇	Λ	Λ̇	ᐳ	ᐳ̇	ᐸ	ᐸ̇	ᐸ̇	Ո	Ո̇
90	Ɔ	‘	’	“	”	•	-	—	ᑐ	™	ᑕ	ᑕ̇	ᑦ	ᑭ	ᑭ̇	ᑯ
a0	NB SP	ᑯ̇	ᑲ	ᑲ̇	ᑳ	ᑵ	ᑵ̇	ᒍ	ᒍ̇	©	ᒪ	ᒪ̇	ᒫ	ᒥ	®	ᒥ̇
b0		ᒧ	ᒧ̇	ᒪ	ᒪ̇	ᒫ		ᓂ	ᓄ	ᓅ	ᓇ	ᓈ	ᓐ	ᓯ	ᓰ	ᓱ
c0	ᓴ	ᓵ	ᓷ	ᓷ̇	ᓸ	ᓕ	ᓖ	ᓗ	ᓘ	ᓚ	ᓛ	ᓭ	ᓬ	ᓮ	ᓯ̇	ᓰ̇
d0	ᔨ	ᔨ̇		ᔪ	ᔪ̇	ᔫ	ᔫ̇	ᔭ	ᔭ̇	ᔮ	ᔾ	ᔾ̇	ᕆ	ᕆ̇	ᕇ	ᕇ̇
e0	ᕈ	ᕈᑭ	ᕈᑭ̇	ᕈᑯ	ᕈᑯ̇	ᕈᑲ	ᕈᑲ̇	ᕈᑳ	ᕈᑵ	ᕈᑵ̇	ᕈᒍ	ᕈᒍ̇	ᕈᒪ	ᕈᒪ̇	ᕈ	ᕈᑦᑭ
f0	ᕈᑦᑭ̇	ᕈᑦᒪ	ᕈᑦᒪ̇	ᕈᑦᒫ	ᕈᑦᒫ̇	ᕈᑦ	ᕗ	ᕘ	ᕙ	ᕚ	ᕛ	ᕜ	ᕝ	H		

Table 2: The Inuit character table that the `inuitscii` ΩTP implements.

```
\ExecuteOptions{nunavut}
\ProcessOptions
```

Each option corresponds to some input encoding. Thus we declare an ΩCP list that can be pushed on the ΩCP stack to become the default input method. Note that since UCS-2 is the default input encoding, we need to leave the input intact when this option is used. This is the reason we load the `id` ΩCP.

Before we proceed with the definition of a number of commands, we need to input the font encoding file.

```
\input litenc.def
```

The encoding file contains only absolutely necessary information:

```
\DeclareFontEncoding{LIT}{}{}
\DeclareFontSubstitution{LIT}{cmr}{m}{n}
\DeclareErrorFont{LIT}{cmr}{m}{n}{10}
```

In the case of the Λ format being built with more than one set of hyphenation patterns, we need a command that can be used to select the hyphenation patterns we wish. We opted to implement a command that is available with the babel package, so that users are familiar with its use:

```
\def\selectlanguage#1{%
 \expandafter
 \ifx\csname l@#1\endcsname\relax%
   \typeout{^^J Error:No hyphenation
     patterns for language #1 loaded,}%
    \typeout{ default hyphenation
              patterns are used.^^J}%
  \language=0%
  \else\language=\csname l@#1\endcsname%
  \fi}
```

The declaration `\inuittext` should be used to permanently change the font encoding and pop the corresponding ΩCP list:

```
\def\inuittext{%
     \fontencoding{LIT}\selectfont%
    \def\encodingdefault{LIT}%
    \selectlanguage{inuit}%
    \pushocplist\InInuitList%
}
```

Now it is easy to implement a new environment that has the same functionality. We invite the reader to try to implement this new environment and to compare his/her implementation with ours.

As we have already explained, the command `\InuitToday` prints the current date in the Inuktitut language. Here is the code:

```
\DeclareRobustCommand{\InuitToday}{{%
   \fontencoding{LIT}\selectfont%
   \number\day\space%
   \ifcase\month%
    \or ^^^^1528^^^^14c4^^^^140a^^^^1546
    \or ^^^^1555^^^^1433^^^^140a^^^^1546
    ..............................
   \fi%
   \number\year}}
```

Note that the Inuktitut letters are typed in using Ω's `^^^^hhhh` notation, where `hhhh` are lowercase hexadecimal digits. With this notation we can specify the code point of any UCS-2 Unicode character, analogous to TEX's `^^hh` notation.

4.2 The Ω Translation Processes

To provide the functionality described above, we designed three ΩTPs: `inuitscii`, `Ninuit2uni` and `Qinuit2uni`. The first of these implements the 8-bit codepage presented in Table 2, while the other two allow users to enter Inuktitut text using the Latin transcription presented in Table 1. The `Ninuit2uni` ΩTP produces Inuktitut text that follows the Anglican orthography, while the `Qinuit2uni` ΩTP produces text that follows the Catholic orthography.

Because `Ninuit2uni` produces a character set that is a superset of the one `Qinuit2uni` produces, we describe only the structure of the first ΩTP. To begin with, we present the input and output sections:

```
input : 1;
output: 2;
```

Ω reads single byte characters and produces two-byte characters. The first thing we must handle in the expressions section is the vowels of the syllabary:

```
expressions:
   ‘i’ ‘i’ => @"1404;
   ‘i’     => @"1403;
   ‘u’ ‘u’ => @"1406;
   ‘u’     => @"1405;
   ‘a’ ‘a’ => @"140B;
   ‘a’     => @"140A;
   ‘h’     => @"157C;
```

Here we see that two consecutive vowels are mapped to one character, which is actually the dotted version of the character that the single vowel is mapped to. Note that if we change the order and try to handle the short vowel first, it will not be possible to handle the long vowel. So one must be very careful when designing ΩTPs.

Now, we will describe how we handle syllables that start with a particular consonant. If the consonant is the last character of the input stream, we simply push its Unicode equivalent to the output stream. If the consonant is not followed by one of the vowels, then we push the character that immediately follows the consonant back to the input stream and push the corresponding Unicode character to the output stream. Finally, depending on the vowel (or vowels) that follow the consonant, we push to the output stream the Unicode character that corresponds to this syllable. Here we show only the case for the consonant p:

```
‘p’ end:           => @"1449;
 ‘p’ ^(‘i’|‘a’|‘u’) => @"1449 <= \2;
‘p’ ‘i’ ‘i’        => @"1432;
‘p’ ‘i’            => @"1431;
‘p’ ‘a’ ‘a’        => @"1439;
‘p’ ‘a’            => @"1438;
‘p’ ‘u’ ‘u’        => @"1434;
‘p’ ‘u’            => @"1433;
```

The other consonants are treated the same way.

The `inuitscii` ΩTP is programmed in a different way. Here we are dealing with an 8-bit code page (namely ISCII) that is essentially an extended ASCII character set. This means that the lower part of the character set will be identical to ASCII and the upper part will contain the Inuktitut letters. So we define an array whose elements are the code points of the Inuktitut characters:

```
tabInuitSCII[@"81] = {
@"1403, @"1404, @"1405, ........
@"1432, @"1433, @"1434, ........
.............................
 @"1672, @"1673, @"1674, ........};
```

Now, we need a way to map the Inuktitut letters to the corresponding Unicode characters. The mapping is not difficult:[1]

```
@"00-@"7F  => \1;
 @"80-@"FF  => #(tabInuitSCII[\1-@"80]);
.           => @"FFFD;
```

Characters that belong to the lower part of the ISCII are mapped to themselves. Characters that belong to the upper part of the ISCII are mapped to a table entry that is located at $c-128$ where c is the ordering number of the Inuktitut letter in the ISCII character set. Note that the hexadecimal number 80 is equal to the decimal 128.

4.3 The fonts

The oinuit package uses virtual fonts that are built around the Computer Modern sans serif font and a PostScript version of the Nunacom TrueType font developed by Nortext (`http://www.nortext.com`), which is redistributed with permission from Nortext. We use the Computer Modern sans serif font because this better matches the Nunacom font we use.

Here we present only the general structure of the Ω virtual property list files that we had to create to allow users to enter UCS-2 encoded text directly. In the beginning of each ΩVP file, we have the identification part and the assigment of the various font dimensions:

```
(FAMILY OINUIT)
 (CODINGSCHEME Unicode Inuit)
(DESIGNSIZE R 10.0)
(FONTDIMEN
   (SLANT R 0.0)
```

[1] After all, in computer science arrays sometimes are treated as functions or, more generally, as mappings.

```
(SPACE R 0.5)
(STRETCH R 0.3)
(SHRINK R 0.1)
(XHEIGHT R 0.583)
(QUAD R 1.0)
)
```

Each virtual font includes the ASCII characters and the characters used in Inuktitut. So the font dimensions are really "average" font dimensions. The two different fonts are introduced with `MAPFONT` definitions:

```
(MAPFONT D 0
   (FONTNAME Inuit)
    (FONTDSIZE R 10.0)
   )
(MAPFONT D 1
   (FONTNAME cmss10)
    (FONTDSIZE R 10.0)
   )
```

Although character entries are quite standard, we present just one so that readers can see what has to be done.

```
(CHARACTER H 0021
   (CHARWD R 0.256)
   (CHARHT R 0.689)
   (CHARDP R 0.004)
   (MAP
       (SELECTFONT D 0)
      (SETCHAR 0 41)
      )
   )
```

It is important to note that we had to create the font `cmssbxo10` in order to have the matching Computer Modern sans serif bold oblique font for the corresponding Nunacom font.

4.4 Hyphenation patterns

Hyphnenating Inuktitut documents written in syllabics is fairly easy because there are no hyphenation rules! However, breakpoints cannot appear before any final consonant (or diacritic signs), except for bigger symbols — such as the symbols for *ng* or *q* — which include a final consonant within the symbol itself.

By default, all these Inuktitut letters have catcode "other", so we set it to "letter". In addition, we set the lowercase codes and the uppercase codes of each symbol. Since there are no uppercase or lowercase letters, we define that the uppercase/lowercase of a symbol is the symbol itself. Here is an example declaration:

```
\catcode‘^^^^1403=11
 \lccode‘^^^^1403=‘^^^^1403
 \uccode‘^^^^1403=‘^^^^1403
```

The fact that we can break a word at any point can be expressed as follows: Given a letter `c`, the pattern `c1` means that it is possible to break a word just after this letter. In addition, the exception is expressed as follows: Given a letter `c`, the pattern `2c` prohibits hyphenation before the letter `c`. Here is an "excerpt" from the patterns declarations:

```
\patterns{%
 ^^^^14031 ^^^^14041 ........
 ............................
 2^^^^1449. 2^^^^1466. ......
}
```

5 Conclusions and future work

We have presented our views regarding package development for Λ, and along these lines we presented the design principles of a particular package. We believe that our work can be used as a starting point for development of a set of widely-accepted principles for the development of Λ packages. This would be particularly useful in the framework of the LaTeX3 project. At any rate, we plan to use our experience to implement a number of other packages that will provide the TeX community with new typesetting capabilities.

6 Acknowledgments

I would like to thank Andrea Tomkins for giving me the right to redistribute the Nunacom font. I also thank Luis-Jacques Dorais, who explained to me the hyphenation rules of the Inuktitut language, and Dimitrios Filippou, who clarified the secrets of transforming these rules into hyphenation patterns suitable for use with TeX/Ω. Finally, thanks to the *TUGboat* reviewers Steve Peter and Karl Berry for clarifying the linguistics section and offering other general suggestions.

References

[1] Kenn Harper. Writing in Inuktitut: An Historical Perspective. Available from `http://www.nlc-bnc.ca/nord/h16-7301-e.html`, September 1983.

[2] Apostolos Syropoulos, Antonis Tsolomitis, and Nick Sofroniou. *Digital Typography Using LaTeX*. Springer-Verlag, New York, N.Y., USA, 2003.

◇ Apostolos Syropoulos
366, 28th October Str.
GR-671 00 Xanthi, Greece
`apostolo@ocean1.ee.duth.gr`

LaTeX News

Issue 14, June 2001

Future releases

We are currently exploring how to best support the very large community of individuals, organisations and enterprises that depend on the robustness and availability of the current standard LaTeX distribution. The results of this may lead to some changes in the regular release schedule and the handling of bug reports during the next year.

New release of Babel (required)

Earlier this year a new release of Babel (3.7) became available. You can read about its new features in

`http://www.ctan.org/tex-archive/macros/latex/required/babel/announce.txt`

One of the bugs that got fixed in this release deals with how labels are handled by LaTeX. Because this part of the kernel is modified by babel, the relevant changes need to be coordinated. Therefore to use Babel with this release of LaTeX you will need to update your version of babel to at least 3.7.

New input encoding latin9

The package inputenc has, thanks to Karsten Tinnefeld, been extended to cover the latin9 input encoding. The ISO-Latin 9 encoding is a useful modern replacement for ISO-Latin 1 that contains a few characters needed for French and Finnish. Of wider interest, it also contains the euro currency sign; this could be the killer argument for many 8-bit texts to use Latin-9 in the future.

According to a Linux manpage, ISO Latin-9 supports Albanian, Basque, Breton, Catalan, Danish, Dutch, English, Estonian, Faroese, Finnish, French, Frisian, Galician, German, Greenlandic, Icelandic, Irish Gaelic, Italian, Latin, Luxemburgish, Norwegian, Portuguese, Rhaeto-Romanic, Scottish Gaelic, Spanish and Swedish. The characters added in latin9 are (in LaTeX notation):

```
\texteuro  \v S  \v s  \v Z  \v z  \OE  \oe  \" Y
```

They displace the following characters from latin1:

```
\textcurrency  \textbrokenbar  \"{}  \'{}  \c{}
\textonequarter  \textonehalf  \textthreequarters
```

New tools

The new package trace provides many commands to control LaTeX's tracing and debugging output, including the excellent new information available with ε-TeX such as the extremely useful tracing of local assignments. You will find it in the tools distribution.

It offers the command `\traceon`, which is similar to `\tracingall` but suppresses uninteresting stuff such as font loading by NFSS (which can go on for pages if you are unlucky). It also offers `\traceoff` to ... guess what! Full details are in the documented source file, `trace.dtx`.

In the base ifthen package we have added the uppercase synonyms `\NOT` `\AND` and `\OR`.

New experimental code

In *LaTeX News 12* we announced some ongoing work towards a 'Designer Interface for LaTeX' and we presented some early results thereof. Since then, at Gutenberg 2000 in Toulouse and TUG 2000 in Oxford, we described a new output routine and an improved method of handling vertical mode material between paragraphs. In combination these support higher quality *automated*[1] page-breaking and page make-up for complex pages—the best yet achieved with TeX!

More recently we have added material to handle the complex front matter requirements of journal articles; this was presented at Gutenberg 2001 in Metz.

A paper describing the new output routine is at

`http://www.latex-project.org/papers/xo-pfloat.pdf`

All code examples and documentation are available at

`http://www.latex-project.org/code/experimental`

This directory has been extended to contain the following.

galley Prototype implementation of the interface for manipulating vertical material in galleys.

xinitials Prototype implementation of the interface for paragraph initials (needs the `galley` package).

xtheorem Contributed example using the `template` package to provide a designer interface for theorem environments.

xor A prototype implementation of the new output routine as described in the `xo-pfloat.pdf` paper.

xfrontm A prototype version of the new font matter interface.

[1] The stress here is on automated!

LaTeX News

Issue 15, December 2003

Anniversary release

Yes, it's now 10 years since the first release in this series and, for Knuthists, this release also contains *Issue 16* !

Meanwhile this *Issue 15* describes the major new features in the current release whilst *Issue 16* looks a little way into the future of LaTeX.

LPPL – new version

Most importantly, there is now a new version, 1.3, of the LaTeX Project Public Licence. Many of you will be thrilled to know that, following the exchange of over 1600 e-mail messages dissecting various aspects of its philosophy such as 'how many angels can appear in the name of a file before it becomes non-free', this version is now officially a DFSG (Debian Free Software Guidelines) approved license. The discussions start at `http://lists.debian.org/debian-legal/2002/debian-legal-200207/threads.html` with high traffic throughout August to October 2002 and further heated discussions starting in April 2003 and concluding around June at `http://lists.debian.org/debian-legal/2003/debian-legal-200306/msg00206.html`.

The important features of the new version are useful clarifications in the wording, and revised procedures for making a change to the Current Maintainer of a package. Special thanks to all those people from Debian Legal who worked constructively with us on this onerous task, especially but not exclusively Jeff Licquia and Branden Robinson.

Small updates to varioref

The English has been corrected in `\reftextbefore` (an incompatible change). There are other extensions such as `\labelformat`, `\Ref`, `\Vref` and `\vpagerefnum`. Some Dutch text has also been changed and two new options added: slovak and slovene.

New and more robust commands

Many of the math mode commands for compound symbols have been made robust and a new robust command has been added: `\nobreakdashes`. This last is a low-level command, borrowed from the amsmath package, for use only before hyphens or dashes. It prevents the line break that is normally allowed after the following sequence of dashes.

Fixing font sizes

The new fix-cm package, by Walter Schmidt, changes the CM font definition (`.fd`) files so that similar design sizes are used in both the `OT1` and `T1` encodings.

Font encodings

A number of options have been added to the textcomp package, enabling only available glyphs to be used. Also, the 'NFSS font families' are now divided into five different groups according to the subset of glyphs each provides from the full collection of symbols in the TS1 encoding. Given sufficient information about a font family textcomp will use this in order to limit the typesetting to those glyphs that are available.

Use of this mechanism has also enhanced `\oldstylenums` to use the current font if possible.

Displaying font tables

With the nfssfont package you can now specify the font to display by giving its 'NFSS classification', rather than needing to know its external font file's name. It is also now possible to generate large collections of font tables in batch mode by providing a suitable input file.

New input encodings

The inputenc package has been extended as follows: macce input encoding (Apple Central European), thanks to Radek Tryc and Marcin Wolinski; cp1257 for Baltic languages; latin10, thanks to Ionel Ciobîcă. The euro symbol has by now been added to several encodings: ansinew, cp1250 and cp1252 (which also has another addition), whilst cp858 adds it to cp850.

Unicode input

Partial, experimental support for text files that use the Unicode encoding form UTF-8 is now provided by the option utf8 for the inputenc package.

The only Unicode text file characters supported by the current version are those based on the most common inputs for glyphs from the small collection of standard LaTeX Latin encodings.

And finally . . . pict2e

The old, non-functional version of this package has been removed as there is now a fully working version from Hubert Gäßlein and Rolf Niepraschk. It is described in *The LaTeX Manual*.

LaTeX News

Issue 16, December 2003

Anniversary news

This anniversary *Issue 16* takes a brief look into the future work of the LaTeX3 Project Team, both short and and longer range. Please let us know if you want to get involved with us in any of this work (see below).

An overview of the 10th Anniversary Release, dated 2003/12/01, is can be found in *Issue 15*.

TLC2: The LaTeX Companion – 2nd edition!

Since you are reading this newsletter, there is a good chance that you, or a friend, has already bought this encyclopedic volume: the incomparable Second Edition of this work that is every LaTeXie's ultimate lucky charm.

If by some chance you have not yet purchased your own copy then get into training, get shopping, and get flexing your muscles (both physical—it's 1100+ pages, and intellectual) by using it to discover masses of invaluable 'insider information' about:

- the latest release of Standard LaTeX;
- over 200 extension packages;
- plus related software and systems.

For more information on this all new (??... OK, not *all*, but over 90%!!), all accurate (we hope!) 10th Anniversary Edition, check out `http://www.awprofessional.com/titles/0201362996`.

Future maintenance

We are currently exploring how best to support the very large and rapidly growing community of individuals, organisations and enterprises that depend on the robustness and availability of the current standard LaTeX distribution. Although we remain firmly resolved not to make changes in the base distribution (the kernel) of Standard LaTeX, there is still much that needs doing to maintain its reliability and utility and to keep up the necessary level of communication with users and supporters. Also, as with all advanced software systems, bugs are still turning up occasionally so some fixes are still essential.

One major impediment to providing adequate service levels in this area is, of course, the difficulties inherent in obtaining the time and commitment of skilled minds—hence the appeal above to anyone interested in getting involved.

LPPL certification

There are still some outstanding diplomatic tasks around the LaTeX Project Public Licence: these include e.g., getting it 'OSF certified' and ensuring that it gains more support and wider use, even in the FSF world where it has long been tolerated.

Use of ε-TeX/pdfTeX

We expect that within the next two years, releases of LaTeX will change modestly in order to run best under an extended TeX engine that contains the ε-TeX primitives, e.g., ε-TeX or pdfTeX. The details of this possible upgrade need further work so we are not making a definite announcement yet.

Although the current release does not *require* ε-TeX features, we certainly recommend using an extended TeX, especially if you need to debug macros.

End of 'autoload' support

As computer systems generally grow in capacity, requirements change and so we believe that the autoload variant of LaTeX is no longer required. Thus, although the code remains it is no longer supported. We hope this does not cause any problems.

New models, new code

In the period 1999–2001 we published many results of our work over the previous decade on the development of new concepts and models for automated typesetting based on TeX as the underlying platform. These can be found at `http://www.latex-project.org/papers/` and `http://www.latex-project.org/code/experimental/`.

Since then a very large proportion of the The Team's efforts have been diverted to provide the core author team for TLC2, which provides over 1000 pages of carefully researched and tested documentation of many aspects of the vast world of LaTeX related software that was developed over that same time period and that continues to grow and improve prodigiously.

Completion of that task ... until TLC3!! ... presents the possibility of getting back to this more exciting development work, or even to more radical work on non-TeX-based models and implementations.

Of course, any such ideas are predicated on our ability to organise (with you, we hope) an efficient but responsive maintenance and support system for Standard LaTeX.

Macros

ednotes — critical edition typesetting with LaTeX

Uwe Lück

1 Overview

1.1 Introduction

For typesetting critical text editions in the traditional manner, using TeX, there are currently three packages available from CTAN: EDMAC, LEDMAC and our ednotes. We list virtues and shortcomings of these three solutions and explain the features and usage of ednotes.

To be sure, there is a fourth package poemscol, available from the CTAN directory `macros/latex/contrib/poemscol`, written by John Burt, especially for critical editions of collections of poems (Burt, 2001). We do not include this package in our comparison — we have not studied it.

We are reporting on version 3.17 of edmac.doc, version 0.51 of ledmac.dtx, and version 1.0 of our ednotes.sty. (We will also report on other files, without listing all their version specifications.)

1.2 Summary of comparisons

Essentially, only either EDMAC or LEDMAC on one side has to be compared with ednotes as the "opponent" on the other side. (We support this claim at the beginning of section 2.3 and undermine it at the same section's end.)

We list a number of tasks that a package for critical editions should accomplish. Some of these tasks are only accomplished by EDMAC and LEDMAC and not by ednotes. On the other hand, as to some solutions that all three packages accomplish the ednotes solution might be considered superior (with respect to the user interface). There even are a few little things which ednotes can do and EDMAC and LEDMAC (at present) cannot.

1.3 Sketch of ednotes features

ednotes provides, firstly, a command `\Anote` such that the input

⟨*lcode*⟩`\Anote{`⟨*lemma*⟩`}{`⟨*note*⟩`}`⟨*rcode*⟩

yields the following output:

- in the main text (of the page or column at which TeX is currently working), printed output is the same as resulting from ⟨*lcode*⟩⟨*lemma*⟩⟨*rcode*⟩.
- ⟨*note*⟩ is printed in the uppermost of all footnote "layers" (of which there may be up to five) of the same page. ⟨*note*⟩ is there preceded with the number(s) of the main text line(s) in which ⟨*lemma*⟩ appears, with a repetition of (a variant of) ⟨*lemma*⟩, and with some separating stuff — see the input in figure 2 for the sample output in figure 1. At the user's choice, some of the line numbers mentioned appear in the margin of the main text.

1 There is nothing special to note in the first
2 line, neither in the second one.

1 first] upper **2** second] lower

Figure 1: Output of critical edition sample

```
\begin{linenumbers}
There is nothing special to note in
the \Anote{first}{upper} line, neither
in the \Anote{second}{lower} one.
\end{linenumbers}
```

Figure 2: Source for critical edition sample

By calling the package with extra options, you can create commands `\Bnote` etc. as well as new footnote layers, and you can choose their style (one common block on each page vs. single blocks — see (T4) below). ⟨*lemma*⟩ may have shapes like

⟨*start lem*⟩`\<`⟨*inner lem*⟩`\>`⟨*end lem*⟩

to indicate what short version of ⟨*lemma*⟩ is to precede the note. There are many facilities to customize appearance of notes. Commands

`\Anotelabel{`⟨*label*⟩`}`, ...

plus

`\donote{`⟨*label*⟩`}{`⟨*note*⟩`}`

vary `\Anote` so that lemmas may overlap. Further facilities allow use of the former commands even in some LaTeX tabular environments.

2 Task(s) and "rival" solutions

2.1 The task(s) of critical edition typesetting

Critical editions are needed in historical text-based work in the arts or sciences when the goal is finding a "definitive" version of a handwritten manuscript or of text that has been edited (in print or in copying by hand) several times. (For this and the following, cf. the exposition of the task in (Burt, 2001), whose author is obviously better informed on the subject than U. L.)

In a critical edition, the "true" text is printed as the main body of a page, and variant readings, remarks, and the like are printed at the bottom of the page. The traditional style of critical editions has the following features, which are thus also the tasks that the packages we discuss here have to handle.

To summarize the main feature in advance: variant readings, remarks and the like (let us call these things '*notes*') do not appear as standard footnotes, as, e.g., LaTeX provides them through `\footnote`. There are no footnote marks for indicating which note comments which passage of the main text. Rather (and here come the single partial tasks):

(T1) (**Marginal line numbers**) Consecutive numbers of the lines of the edited text are printed in the margin.

(T2) (**Keying**) To which passage of the main text a note refers is indicated by preceding the note with the line number and a (partial) repetition of that passage (which scholars call '*lemma*' and which often is just a single word).

(T3) (**Multiple notes series**) Typically, there are at least two separate kinds of notes — such as variant readings, text-critical notes, and testimonia — which use different layers at the bottom of each page.

(T4) (**Formatting notes compactly**) Notes of certain typical kinds are so short that much space would be wasted if each note was printed on its own line(s) (as would happen with a LaTeX `\footnote`). Rather, all the notes of a page belonging to one kind ("layer", "series") are arranged in (a) a single paragraph (**block formatting**) or even in (b) one layer of two or three columns (**columnar formatting**).

The above tasks are "musts"; a package not accomplishing them would be of no practical utility for critical editions. There are other goals which some authors would like or even urgently need, but which other authors do not require. Such are:

(X1) (**Cross-references to lines**) Neither Plain TeX nor LaTeX provide a mechanism for cross-referring to the *line number(s)* of a certain passage. This might be needed especially for commentary paragraphs *between* edited texts (e.g., if they are letters and need a long exhibition of background). Indeed, such a mechanism could be used for accomplishing task (T2).

(X2) (**Line numbering switches**) Depending on how long edited texts are and whether you need main text commentary surrounding them, it must be possible to switch numbering of lines on and off, or to restart numbering. Moreover, authors should be able to choose whether line numbers appear on the left or on the right side of the main text. It might also be desirable to choose whether all line numbers or whether, e.g., only every fifth linenumber is printed in the margin.

(X3) Editing *plays* often requires treatment of "**sublines**" and their numbering. As well, additional features for editing *poetry* are valuable.

(X4) (**Columnar notes formatting**) We repeat the problem (b) of arranging notes in columns at the bottom of the page from task (T4), since when when *block formatting* (T4) (a) notes, there is no longer a vital need for (b).

(X5) While footnotes may be appropriate for some kinds of notes, **endnotes** might be more appropriate in other cases.

(X6) (**Lemma abbreviations**) When the lemma is rather long, it should be displayed partially only preceding the note at the bottom of the page.

(X7) (a) **Nested** or even (b) **overlapping lemmas** may sometimes be needed.

(X8) (**Count word occurrences**) The "referring" feature (T2) is ambiguous if the lemma word occurs more than once in the given line. Traditionally this problem has been handled with an index n in the repetition of the lemma word preceding the note when the note refers to the n^{th} occurrence of the word in the line. Doing this manually is quite tedious, and so TeX macros to automate this job are often asked for.

(X9) Publishers like "**crop marks**" on camera-ready copies.

(X10) (**Lemmas in bad places**) Some features seeming very natural to TeX-laymen turn out to somewhat "resist" implementation (essentially due to some weaknesses of the TeX program). One example is the case of (a) lemmas inside **math expressions** — especially in (equation) displays; another is (b) lemmas in **tables**.

2.2 History (and availability) of "rivals": EDMAC, LEDMAC, ednotes

Starting in 1987, John Lavagnino and Dominik Wujastyk wrote TeX macros for critical editions, originally of plays. This work terminated in 1996 with version 3.17 of the EDMAC package. Many researchers have, for their professional publications, used these macros by now, even for Arab and Sanskrit editions. Its manual and documentation are available as a beautiful book (Lavagnino and Wujastyk, 1996) from TUG; it also tells more about the

history and usage of EDMAC. An overview appeared in (Lavagnino and Wujastyk, 1990). An EDMAC software distribution is freely available from CTAN, in `macros/plain/contrib/edmac`. And finally, Dominik Wujastyk maintains a beautiful *home page* for EDMAC at

`http://www.ucl.ac.uk/~ucgadkw/edmac/`

from which also some of the packages mentioned here can be downloaded. This web page also reports on alternatives to EDMAC for critical edition typesetting.

So all seemed to be happy. However, ...

When John and Dominik started their EDMAC project, Leslie Lamport's LaTeX format for TeX already had been born and was spreading widely among TeX users. By contrast, EDMAC had been written for the Plain TeX format, as described in Donald Knuth's TeXbook (Knuth, 1996). EDMAC is essentially incompatible with LaTeX (cf. section 2.3 below). It seems that nowadays most TeX users work with the LaTeX format, while Plain TeX is only used by a few exotics for, say, the history of science or music. The historians were tied to Plain TeX because they could not live without EDMAC.

In late 2002, Christian Tapp hired U. L. for a research project at the Chair for History of Science at the University of Munich. Christian expressed his sorrow that he needed TeX macros for his critical editions in the project, while being very adverse to 'learning Plain TeX' beyond LaTeX just to be able to use EDMAC. Uwe expressed his joy in writing TeX macros. Christian knew from many of his colleagues at the chair that they found it a nuisance that there was nothing resembling EDMAC that, at the same time, was compatible with LaTeX. This was the birth of our ednotes, which is now available from CTAN in directory `macros/latex/contrib/ednotes`. (Christian devised functions, and U. L. typed the definitions.) So it seemed that from many's lamenting and Uwe's joy with TeX macros (and knowledge of some LaTeX internals) much more happiness emerged than there had been before in the EDMAC era. However, ...

One item of bad news is that one of the most severe bugs was fixed only in January 2004; until then we did not really dare to claim that the package worked. (However, we could help some test users with the problems they had with ednotes.) And more testing may be needed to see whether this situation has essentially improved. (However, at least Christian indeed worked with ednotes, using some awkward tricks to circumvent the bugs, or, at times, just enduring the bugs.)

Another bad news is that there are still some things that EDMAC can do and ednotes cannot, see section 2.3. And there are still things which *no* known package does as intended, see section 2.4.

Even the start was quite bad: Uwe saw that doing something like EDMAC in LaTeX needed a lot of knowledge of TeX and LaTeX internals and a lot of work. We were near to giving up. At this point, Christian luckily found two packages each of which relieved almost half of our burden. Stephan Böttcher's lineno.sty does all the work concerning line numbering — tasks (T1), (X1), and (X2) from the above. Alexander Rozhenko's manyfoot does all the work concerning multiple series of footnotes, some of which may be block formatted — tasks (T3) and (T4) (a). We only needed to add a user interface that would pass the author's wishes to the two packages in a nice way. Indeed, we did not try to emulate EDMAC, but thought of an even somewhat smarter user interface than EDMAC's — concerning overlapping lemmas (X7) (b), for example.

Finally, an issue arose when Peter Wilson came forward in March 2003 with apologies for not having known about our project (which, by then, had been announced on the EDMAC home page) and for developing a(n almost entirely) faithful copy of EDMAC for use with LaTeX. He called it 'LEDMAC'; it has been freely available from CTAN, `macros/latex/contrib/ledmac`, since a few days later.

The latter problem consisted in Peter, Christian and Uwe being afraid that all their work on TeX macros for critical editions had been in vain. At this point (re-)appeared Dominik Wujastyk on the scene, bringing peace by encouraging all of us, saying that it would be good if users had a choice. Indeed, LEDMAC and ednotes have different user interfaces and are implemented through quite different mechanisms. — Since then, all have seemed to be relatively happy. (Don't forget the bugs we had for such a long time.)

2.3 When to use which package

In this subsection we list virtues and shortcomings of the three solutions introduced above, hoping to give useful advice for readers pondering the question of which solution they should adopt. (Originally, Dominik Wujastyk suggested that it would be nice if such a comparison were offered in the documentation of both LEDMAC and ednotes.)

EDMAC incompatible with LaTeX? Throughout this paper the reader will find claims that EDMAC is not compatible with LaTeX. However, these claims are somewhat inspired by *The TeXbook*'s

(Knuth, 1996, p. vii) didactic method of temporary lying. At this point we try to stay closer to the whole truth.

(1) The EDMAC bundle (i.e., the content of the CTAN folder .../edmac) actually provides a LaTeX package edmacfss.sty for loading EDMAC under LaTeX. However, the purpose of this is primarily to provide LaTeX 2ε's (Frank Mittelbach's and Rainer Schöpf's) New Font Selection Scheme (NFSS) for use with EDMAC. So you run LaTeX on some document file which contains the command

```
\usepackage{edmacfss}
```

and use EDMAC commands in the document body. A considerable portion of LaTeX beyond NFSS will then at the same time work — but another considerable portion will not. First of all, edmacfss.sty reinaugurates the Plain TeX meaning of `\end` — so none of LaTeX's "environments" are available. Another large portion is everything concerning floats (including `\marginpar`s) and page layout — since EDMAC overwrites LaTeX's `\output` routine.

(2) One might just load edmac.doc (or a docstripped version of it) to use EDMAC under LaTeX. This would at least preserve the LaTeX meaning of `\end` and thus LaTeX environments. But the other compatibility problems named in (1) above will remain.

To conclude: You may try using EDMAC with format LaTeX. You may luckily succeed, using only a certain portion of LaTeX. Drawing *exactly* the line between the portions of LaTeX compatible with EDMAC and the portions incompatible might be helpful, but we don't try here; and for the sake of simplicity we will go on to claim that EDMAC is incompatible with LaTeX. — A variation of this theme is the content of ed-nfss.txt which comes along with EDMAC (in CTAN folder .../edmac as well as on the EDMAC home page).

Plain TeX or LaTeX? We assume throughout (relying on Peter Wilson's information) that LEDMAC is (as intended) a faithful copy (port) of EDMAC into LaTeX. This means that, except for a few command names, the functionality and user interface of EDMAC and LEDMAC are the same. (We will report below that LEDMAC has become even more powerful than EDMAC, but this need not bother us for the next few paragraphs.) For convenience, we therefore stipulate a hypothetical, imaginary being called "(L)EDMAC" which is one of EDMAC or LEDMAC with no definite decision as to which of the two it is (cf. Schrödinger's cat).

In the following, we will just compare (L)EDMAC with ednotes. If the reader sees that she needs ednotes, not (L)EDMAC, she is bound to use LaTeX, not Plain TeX (ednotes really needs LaTeX, there is no didactical lie at this point). (If the question arises, 'LaTeX' will mean LaTeX 2ε rather than LaTeX 2.09. Our goal was compatibility with LaTeX 2ε, while we have not investigated which of the macros would work with LaTeX 2.09. Indeed, however, the version of LEDMAC that we have scrutinized needs a very recent version of LaTeX 2ε.) If she, by contrast, rather needs (L)EDMAC, it is her personal choice between EDMAC with Plain TeX and LEDMAC with LaTeX.

	(L)EDMAC	ednotes
experience	+(+)	(+)
documentation	++	(+)
(T1) – (T3) "*basics*"	+	+
(T4) "*short notes*"	+	(+)
(X1) *cross-refer to lines*	+	+
(X2) *number switches*	+	+
(X3) *sub-lines, poetry*	+(+)	−
(X4) *columnar notes*	+	−
(X5) *endnotes*	+	−
(X6) *lemma substitutes*	+	++
(X7) (a) *nested lemmas*	+	+
(X7) (b) *overlapping lemmas*	(+)	++
(X8) *count occurrences*	−	(+)
(X9) *crop marks*	(+)	(−)
(X10) (a) *math mode*	(+)	(+)
(X10) (b) *tables*	+	+(+)

Table 1: Performance of (L)EDMAC vs. ednotes

Comparing ednotes to (L)EDMAC. We first refer the reader to table 1 for an overview of comparing ednotes to (L)EDMAC. What the signs (and parentheses) mean will be clear ('+' for implemented, '−' for not implemented, etc.), and we will soon express their meanings in ordinary words and in some detail below. Tags (T1) etc. and (X1) etc., of course, refer to the list of tasks in section 2.1. Concerning the final "score", the reader will immediately observe that there is only one minus for (L)EDMAC while there are several for ednotes, but she should not overlook that, according to the table, ednotes is superior to (L)EDMAC in some respects. This, we hope, compensates for some missing features.

Moreover, this comparison should be considered a "snapshot" only. To be sure, John Lavagnino and Dominik Wujastyk seem to have stopped their work on EDMAC many years ago. By contrast, Peter

Wilson has increased LEDMAC's functionality still this year and might continue doing so. ednotes' authors can conceive of removing some minus signs from their column; however, their capacities and eagerness are limited — but perhaps someone else will do the jobs!? — Inspired by David Kastrup, we remind the reader here explicitly and unashamedly that writing/extending TEX macro packages may be a question of money.

Most of the details of ednotes will be explained in sections 3 and 4 of the article. Here we attempt to compare ednotes to (L)EDMAC *without* giving the exact specifications of corresponding ednotes and (L)EDMAC features. So we will promise that some features of ednotes are superior to their (L)EDMAC counterpart, while the promises are kept only later. However, we will partially anticipate the presentation of the ednotes features so that the reader can make up her mind already through seeing the comparison.

We now simply work ourselves through table 1.

Experience: As told above, at least EDMAC has been used for many years by scholars for many professional publications. Experience with EDMAC transfers to LEDMAC. — By contrast, ednotes is very young, and we know of very few users. Christian Tapp uses it, and we know of three other users, using ednotes for their doctoral dissertations or other professional work. Other people have received an ednotes distribution, but we do not know whether they actually use it.

Documentation: As told above, there is beautiful documentation of EDMAC, available as a book. It is a user manual and at the same time documentation of the implementation. For LEDMAC, Peter Wilson has turned edmac.doc (which is the source code of the EDMAC book) into ledmac.dtx, which is thus a printable user manual and implementation documentation for LEDMAC at the same time. — By contrast, only each the ednotes source files (see a few of them in section 4) carry some user instructions and little explanation of implementation, little of which is printable at this point. The present article will exhibit, as printed, part of the user instructions and almost no hints on implementation. Nevertheless, the packages manyfoot and lineno on which ednotes rests have printable documentations.

"Basics" (T1) – (T4); (X4): Line numbering, keying, multiple layers of footnotes, and compact formatting of notes are provided by both (L)EDMAC and ednotes. ednotes, however, offers block formatting of notes only, not columnar formatting. Moreover, there is a difference in the user interface for doing these things which we describe in section 3. It depends on the kind of work and on person taste which of the interfaces is nicer.

Cross-referring to lines (X1), line numbering switches (X2): (L)EDMAC and ednotes offer the same features in these respects, only the command names differ.

(X3) – (X5): sub-lineation, columnar formatting notes, endnotes, and editing poetry are provided by (L)EDMAC, while not by ednotes. More precisely, poetry is covered by LEDMAC, while Wayne Sullivan's EDSTANZA enhances EDMAC for this purpose.

Lemma tricks (X6) and (X7): Both (L)EDMAC and ednotes support nested lemmas. Moreover, (L)EDMAC offers facilities to change (*i*) the lemma tag preceding the note as compared with the whole lemma in the edited text (`\lemma`) and (*ii*) the line numbers preceding the note (`\linenum`). These facilities could be used to handle overlapping lemmas (indicating boundary line numbers "*manually*", i.e., the user must *know* them "in advance", cf. section 2.3 of edmac.doc and section 3.7 below). ednotes treats abbreviating or replacing lemmas as well as overlapping lemmas with different user interfaces (to be described in sections 3.6f.) — less tricky and more perspicuous, we hope.

(X8) — Counting word occurrences in a line automatically is not enabled by (L)EDMAC at all, while ednotes provides a halfway solution, to be described in section 3.10, where TEX and the user "share" the job.

Crop marks (X9) are available in both alternatives. EDMAC provides them with its own macros. Under LATEX, crop marks are available from, e.g., crop.sty (generated from crop.dtx and crop.ins), the latter available for download at CTAN path `macros/latex/contrib/crop` — search for similar packages on CTAN with the term '`crop`'. In this respect crop marks are available with LEDMAC and ednotes in the same way. (It is difficult to express this situation in the table entries for (X9) properly.)

Difficult positions (X10) — math mode, tables: ednotes offers devices for lemmas in LATEX tables (like `tabular` and and `longtable`). Very recently, moreover, we found modifications that essentially overcome the math mode problem (see the `mathnotes` option described in the package for details). — EDMAC has been augmented by a package tabmac.tex, maintained by Herbert Breger and Nora Gädecke and available from CTAN path `macros/plain/contrib/edmac`. This package offers some facilities for building tables and critical editing of

them using EDMAC. tabmac offers some devices which do not even exist in LaTeX. It even can be used to edit displayed equations (or other math lines). — Peter Wilson has incorporated tabmac into his LEDMAC, with English command names instead of the German ones from tabmac. At present, however, there is only a German user manual (tabm11dc.dvi) for the original German version of tabmac. There is no user manual for its English LEDMAC version. — (L)EDMAC has offered two further devices ("line number substitutes", "lemma substitutes") which could be used to cope with math text.

Further differences concerning items not listed in table 1:

(L)EDMAC and ednotes differ in implementation, viz., use of auxiliary files. However, this seems not to have any practical effects nowadays. On very old machines, (L)EDMAC might be slower than ednotes, while ednotes might cause memory overflow with small TeX versions and many notes.

Meanwhile, LEDMAC — having originally been a LaTeX port of EDMAC — has grown in functionality beyond EDMAC. We have already reported the tabmac functionality incorporated in LEDMAC. Among other features that LEDMAC adds to EDMAC's functionality are: (*i*) indexing by line as well as by page; (*ii*) the functionality of Wayne Sullivan's EDSTANZA for editing a certain kind of verse; (*iii*) a minipage-like environment — even breaking across pages — so that notes appear immediately at its end (instead of at the bottom of the page), which is useful for collections of short edited pieces (letters, e.g.); (*iv*) "sidenotes" (however, since `\marginpar` works with ednotes, it would not be too difficult to add sidenotes as well); (*v*) "familiar" *numbered* footnotes (which exist under ednotes due to the underlying manyfoot package).

On the other hand, Alexander Rozhenko's manyfoot (which ednotes loads) supports different styles of footnote rules, depending on which layers of notes they separate from each other.

2.4 Tasks not accomplished by any package

This may be the right place to point out some shortcomings that all the packages have *in common*.

- If tables contain entries consisting of whole multi-line paragraphs of running text, tricks like the above-mentioned may help in some situations, but there is no user-friendly way to refer to single lines of such paragraphs. Usually table *rows* are numbered, not these "sub-lines". More precisely:

  (*i*) We are sure concerning ednotes. The usual way in LaTeX of producing such paragraph entries is using `p{`⟨*dimen*⟩`}` in the table preamble, which works like `\parbox`es as table entries. The entries in the corresponding column are then single boxes. From ednotes' view, such a box is just a part of a line, ednotes cannot "see" the "sub-lines". The lineno package which is loaded by ednotes (see next section) offers a trick for numbering these "internal" lines and another trick for referring to them, but this is not nice. More generally (perhaps there is some alternative to these `\parbox`es?) such a paragraph entry must first be typeset in a vertical box. In order for each line of this paragraph to be viewed as a part of a line that ednotes can recognize, it would be necessary for the lines of the paragraph to be unpacked from the vertical box and somehow rearranged. ((L)EDMAC and parallel.sty do something like this, but it does not help for ednotes.) There just are no macros for doing this.

  (*ii*) We do not know definitively for (L)EDMAC, and we could not obtain a definite answer from its experts. However, we are convinced that it does not fare better. The situation of first typesetting the entry in a vertical box etc. is essentially the same, and when we scan the commands that EDMAC, tabmac, and LEDMAC offer, we are unable to find one which could do the unpacking and rearranging. With EDMAC and tabmac, there is not even a macro resembling LaTeX's `\parbox`.
- The packages are not compatible with the parallel package which would help for displaying translations.
- None of the packages can handle footnotes in the text to be edited.

Solving these problems would require mechanisms that differ drastically from the present ones (cf. remarks in ledmac.dtx concerning parallel — "a very different implementation for the functionality of parallel seems to be necessary so that line numbering is possible"). Something similar holds for the ensuing problem:

- All the devices for *block formatting* notes in all the packages (all deriving from *The TeXbook*) share the following problem: TeX decides on page breaking considering the heights of the footnotes. All the former macros estimate the height of the final block from the horizontal lengths of notes. So, e.g., there may be four footnote blocks, and the macros tell TeX that each is 2.25 `\baselineskip`s high (because in a very wide box, the notes form a line of two

and a quarter \columnwidths). So TEX reserves 4 times 2.25 \baselineskips of vertical space for all the notes on the page, i.e., 9 \baselineskips. In reality, however, if a note block does not fit into two lines, it needs three of them. So actually the four note blocks need 12 \baselineskips. This discrepancy of 9 vs. 12 \baselineskips may let the notes hang too deeply on the page or even let them overlap with the main text.

Therefore John Lavagnino (co-author of EDMAC) suggested (January 2003) a mechanism very different from these common ones to us — typeset the whole note block for measuring at each note insertion.

David Kastrup suggested that this approach is hopeless and informs us that rather the bigfoot package on which he is presently working solves the problem (a report on David Kastrup's work on critical editions is (Kastrup, 2004)). bigfoot is, in the long run, intended to be a replacement for manyfoot, overcoming the latter's shortcomings; however, replacing manyfoot by bigfoot in ednotes does not work at present.

Recently both LEDMAC (\footfudgefiddle) and manyfoot (\ExtraParaSkip, and thus ednotes) have been enhanced by interim remedies for this problem. (Let us remark again that a proper solution as indicated may depend on money!)

Finally, as remarked above:

- Task (X8), that of counting word occurrences, has no fully automated solution.

3 How to use ednotes

We now turn from comparisons between EDMAC, LEDMAC, and ednotes to a more detailed description of ednotes. The present section describes the *commands* that ednotes (or sometimes lineno) offers.

3.1 Line numbering

The edited text whose lines are to be numbered and to which notes are to refer must be preceded by \linenumbers or must be enclosed in

```
\begin{linenumbers}
...
\end{linenumbers}
```

(see figures 2 and 1 again for an example). These and other commands for task (X2) are provided by Stephan Böttcher's lineno.sty, to whose source documentation (lineno.tex) we hereby refer. The user manual (ulineno.tex) is not quite up-to-date, but the instructions in the comment lines of lineno.sty are easily understandable — see especially the list below \endinput.

There is a bunch of package options for lineno.sty. You can call them as options for ednotes.sty — while their effects are explained in the documentation of lineno.sty. E.g., if you want the modulo feature of lineno (for printing in the margin only the line numbers which are divisible by 5), include modulo in the ednotes package options, as in

```
\usepackage[modulo,...]{ednotes}
```

You may also find the

\linelabel and \lineref

commands from lineno useful to refer to lines of the edited text without using the procedure for notes that ednotes provides. Even

\pageref{⟨*label*⟩}

works with \linelabel{⟨*label*⟩}.

In recent versions, lineno provides a command

\firstlinenumber

by which you can determine which line gets the first visible number attached to it. E.g., if you want to number lines 1, 3, 5, etc., type

```
\modulolinenumbers[2]
\firstlinenumber{1}
```

Without the last command, line numbers 2, 4, 6, etc. would be printed.

3.2 Footnotes

ednotes provides (at your choice) up to five kinds ("layers") of notes. It is your choice which of the five are installed and which of the two available formats of footnotes they will have. ednotes.sty has package options Apara (which is the default option), Aplain, Bpara, Bplain, ..., Epara, Eplain. Whenever you choose the 'para' version, the corresponding "layer" will be *block formatted*. If you choose 'plain' instead of 'para', notes of that layer will be formatted just as it would happen ordinarily in LATEX, every note starting an own line.

Moreover, if you choose the "*block formatting*" style for one of your footnote layers, you can additionally choose whether each block of notes should start with an indent or not. You don't have to do anything if you want the indent; to omit the indent, include para* as a package option:

```
\usepackage[para*,...]{ednotes}
```

ednotes passes package options and commands for inserting notes to the underlying manyfoot package. In general, you need not know anything about the commands that the latter package provides.

However, Alexander Rozhenko has (kindly on Christian Tapp's request) extended his manyfoot with customizing features so you can specify the existence and style of rules between certain footnote layers — if you know how ednotes and manyfoot work together. A documentation file manyfoot.dtx for manyfoot is available from CTAN.

3.3 Keying notes to lemmas — basics

We now turn to the basic features of ednotes.

Recall from section 1.3 that ednotes provides a command `\Anote` such that

`\Anote{`⟨*lemma*⟩`}{`⟨*note*⟩`}`

keys ⟨*note*⟩ to the occurence of ⟨*lemma*⟩ at the place of that `\Anote` in main text. The package options `Aplain`, `Bpara`, ..., `Eplain` mentioned in section 3.2 above make the analogous commands `\Bnote`, ..., `\Enote` available. E.g., options `Bpara` and `Bplain` make `\Bnote` available, and

`\Bnote{`⟨*lemma*⟩`}{`⟨*note*⟩`}`

will send ⟨*note*⟩ into the footnote layer below that of `\Anote`. Anything we say about `\Anote` holds for the analogous commands obtained by these package options.

We use this occasion to emphasize that — as so often — at least two, and usually three, TEX runs are required to get the line number references right.

3.4 Nesting lemmas

In, e.g.,

`\Anote{`⟨*lemma1*⟩`}{`⟨*note1*⟩`}`

⟨*lemma1*⟩ may *contain* a nested note at the same or different level, e.g.,

`\Bnote{`⟨*lemma2*⟩`}{`⟨*note2*⟩`}`

— cf. figures 3 and 4 (where `\Anote` is used instead of `\Bnote`). ((L)EDMAC works similarly. — Here and in future examples we omit `linenumbers` which must appear somewhere according to sections 1.3 and 3.1.)

The same lemma may be used for notes of different kinds, e.g., in

`\Anote{\Bnote{`⟨*lem*⟩`}{`⟨*nB*⟩`}}{`⟨*nA*⟩`}`

```
\Anote{See \Anote{the}{inner}
       sample}{outer}.
```

Figure 3: Code for nesting sample

3.5 Another comparison with (L)EDMAC

The previous example situations offer an occasion for one comparison — to which we alluded earlier — of the user interfaces of (L)EDMAC vs. ednotes. (We choose LEDMAC for examples, EDMAC would just use a slightly different syntax.)

(*i*) `\Anote{`⟨*lem*⟩`}{`⟨*note*⟩`}` of ednotes has the same effect as

`\edtext{`⟨*lem*⟩`}{\Afootnote{`⟨*note*⟩`}}`

has under LEDMAC.

(*ii*) `\Anote{\Bnote{`⟨*lem*⟩`}{`⟨*nB*⟩`}}{`⟨*nA*⟩`}` in ednotes has the same effect as

```
\edtext{⟨lem⟩}%
  {\Afootnote{⟨nA⟩}\Bfootnote{⟨nB⟩}}
```

with LEDMAC.

What do these examples teach us?

- ednotes needs typing of one command name less than (L)EDMAC. However, this can be changed by some simple definitions under (L)EDMAC. With LEDMAC, e.g.:

  ```
  \newcommand*{\Anote}[2]{%
    \edtext{#1}{\Afootnote{#2}}}
  ```

  — so you get the ednotes syntax with (L)EDMAC.
- On the other hand, (L)EDMAC syntax looks better adapted than ednotes' when notes of more than one kind refer to the same lemma.

So the basic syntax may look like an important aspect *prima facie* when you choose between (L)EDMAC and ednotes — but it is *not*. Rather, it is a point in favour of (L)EDMAC. Cf. sections 'The apparatus' and 'Marking text for notes' of the (L)EDMAC documentation. The advantages of ednotes' user interface will be described later.

3.6 Short lemma substitute preceding note

In

`\Anote{`⟨*lemma*⟩`}{`⟨*note*⟩`}`

⟨*lemma*⟩ may appear as ⟨*l1*⟩`\<`⟨*l2*⟩`\>`⟨*l3*⟩ (or this sequence may end earlier: ⟨*l1*⟩`\<`⟨*l2*⟩, e.g.). The lemma tag preceding the note then has form ⟨*l1*⟩⟨*ell*⟩⟨*l3*⟩ where ⟨*ell*⟩ is some ellipsis mark, as explained below, while in the main text just ⟨*l1*⟩⟨*l2*⟩⟨*l3*⟩ appears. The result is that you don't type anything twice, as is the case with EDMAC's `\lemma`.) Figures 5 and 6 exhibit an example.

What appears between ⟨*l1*⟩ and ⟨*l3*⟩ in the lemma tag is customizable for the whole document

1 See the sample.

1 See the sample] outer **1** the] inner

Figure 4: Output of nesting sample

```
This is
\Anote{a \<somewhat long\> lemma}
      {no problem}.
```

Figure 5: Code for ellipsis sample

This is a somewhat long lemma.[1]

[1] a ... lemma] no problem

Figure 6: Output of ellipsis sample

(cf. `\renewcommand` example below). There is a *local* customization possible as well. In that sequence

⟨*l1*⟩`\<`⟨*l2*⟩`\>`⟨*l3*⟩

the ⟨*l2*⟩ may appear as

`<`⟨*ellipsis*⟩`>`⟨*ll*⟩

The tag preceding the note will then be

⟨*l1*⟩⟨*ellipsis*⟩⟨*l3*⟩

while the main text will be

⟨*l1*⟩⟨*ll*⟩⟨*l3*⟩

You may even let ⟨*l1*⟩ be empty.

For the ellipsis, we propose a new symbol `\textsymmdots` which differs from `\dots` in having no space on the right hand side, so the dots can appear really symmetrically between ⟨*l1*⟩ and ⟨*l3*⟩. `\textsymmdots` is the default ellipsis, i.e., if in

⟨*l1*⟩`\<`⟨*l2*⟩`\>`⟨*l3*⟩

'`\<`' is not followed immediately by '`<`', it yields the same output as

⟨*l1*⟩`\<<\textsymmdots>`⟨*l2*⟩`\>`⟨*l3*⟩

does (look at figures 5 and 6 again).

Note, in figures 5 and 6, the blank space before '`\<`' and the one after '`\>`'. These blank spaces are needed for some space before and after the ellipsis dots yielded by `\textsymmdots`. These dots would look quite bad if they were not surrounded by any spaces. Smaller spaces would do, they should at least be `\thinspace` (— we feel). So you see that we have decided that the user should care for these spaces. However, the user can change this feature by, e.g.,

```
\renewcommand{\lemmaellipsis}{%
  \thinspace\textsymmdots\thinspace}
```

in the document preamble (after the `\usepackage` line for ednotes), so she can move the blank spaces into the code between '`\<`' and '`\>`'. Note as well that if ⟨*l1*⟩ and ⟨*l2*⟩ are separate words, you *must* type a blank space either before *or* after '`\<`', otherwise they would appear as if they were parts of a *single* word ⟨*l1*⟩⟨*l2*⟩[...] in main text. Something analogous holds for `\>`. If the lemma is a single long word ⟨*l1*⟩⟨*l2*⟩⟨*l3*⟩, of which only ⟨*l1*⟩ and ⟨*l3*⟩ are to precede the note, it should be typed something like this, without full spaces:

`\Anote{`⟨*l1*⟩`\<<\thinspace\textsymmdots`
`\thinspace>`⟨*l2*⟩`\>`⟨*l3*⟩`}{`⟨*note*⟩`}`

(It might be preferable to introduce an abbreviation with `\newcommand`.)

3.7 Overlapping lemmas

`\Anotelabel{`⟨*label*⟩`}`⟨*lemma*⟩`%`
`\donote{`⟨*label*⟩`}{`⟨*note*⟩`}`

works just like

`\Anote{`⟨*lemma*⟩`}{`⟨*note*⟩`}`

— however, by using suitable ⟨*label*⟩s, you can indicate which of overlapping lemmas begins and ends where; look at figures 7 and 8.

Note that the second command is only `\donote`, not `\Adonote`. Beware as well the blank spaces which line breaks may cause, unless you elide them with the comment mark '`%`'. Usually however, you will rarely be forced into such a situation. We are in such a situation here because the present column width enforces so many line breaks.

```
\Anotelabel{l1}Observe
\Anotelabel{l2}this%
\donote{l1}{Look at this}
sample\donote{l2}{the present sample}.
```

Figure 7: Code for overlap sample

Observe this sample.[1]

[1] Observe this] Look at this [1] this sample] the present sample

Figure 8: Output of overlap sample

In ⟨*lemma*⟩,

`\pause{`⟨*label*⟩`}` and `\resume{`⟨*label*⟩`}`

act analogously to `\<` and `\>` above for lemma substitutes, and

`\pause{`⟨*label*⟩`}<`⟨*ellipsis*⟩`>`

employs your own ⟨*ellipsis*⟩ for the ellipsis. ⟨*lemma*⟩ may contain `\Anote` and the other way round (in some way).

3.8 Further items

We do not deliver a complete user manual here. Let us just note that there are various possibilities to customize the appearence of the note and what

precedes it. The easiest one is perhaps redefining `\Anote` into something like `\variant` etc. The complete instructions can be read in ednotes.sty.

3.9 Editing tables

For critical editing of tables, ednotes offers the options `edtable` and `longtable`. The first one defines an environment `edtable`; the second redefines `longtable` from David Carlisle's longtable.sty (belonging to the Standard LaTeX Tools Bundle).

Environment `edtable`**:**

`\begin{edtable}{`⟨*tabenv*⟩`}[`⟨*pos*⟩`]{`⟨*tmp*⟩`}`
...
`\end{edtable}`

works like

`\begin{`⟨*tabenv*⟩`}[`⟨*pos*⟩`]{`⟨*tmp*⟩`}`
...
`\end{`⟨*tabenv*⟩`}`

— where the first line is meant to be the standard starting line of some LaTeX tabular environment; i.e., ⟨*tabenv*⟩ may be `tabular` or the like, ⟨*pos*⟩ is the positioning argument, and ⟨*tmp*⟩ determines the form of each `tabular` line. The only difference is that you can use ednotes commands within an `edtable`.

Option `longtable`**:** With this ednotes package option, you can use ednotes commands within `longtable` environments — provided the latter are in `linenumber` or like environments. Outside such an environment, `longtable` environments work without any change.

3.10 Multiple occurrences of lemma word

Here we turn to task (X8).

Recall the problem from section 2.1: Sometimes the lemma word occurs more than once in its line. Imagine, e.g., you are editing a Latin text with a line in which the word 'et' occurs three times, and you want to key a note to its second occurrence. The traditional way to handle this situation is to supply the lemma tag preceding the note with an index '2'.

Now, it is rather tedious work to check after printing how often each lemma word occurs earlier in its line. It would be nice if this could be done automatically. However, this would be a very tedious labour for the *macro programmer*. (And perhaps some part of the job would better be done by a program other than TeX in the manner of, e.g., makeindex.)

Only this year we have offered a halfway solution for this job (this was tedious enough programming labour). 'Halfway' means that there remains a job for the author/user. This job is the following: if you are typing some

`\Anote{`⟨*word*⟩`}{`⟨*note*⟩`}`

look some words back for other occurrences of ⟨*word*⟩. Each occurrence which is near enough to your

`\Anote{`⟨*word*⟩`}{`⟨*note*⟩`}`

so that it *might* be printed in the same line should then be made an argument of the command

`\countword`

and so should the occurrence of ⟨*word*⟩ which is the first argument of `\Anote`. Like this:

`\countword{`⟨*word*⟩`}`...
`\Anote{\countword{`⟨*word*⟩`}}{`⟨*note*⟩`}`

The reader may feel cheated by this kind of "solution". However, think of a situation where `\Anote{et}` is preceded by four occurrences of 'et' nearby. `\countword` then saves you from counting how many of these occurrences occur indeed in the same line as the *lemma* 'et'. And this will be even more helpful when you change text width or insert some text before the lemma. Moreover, we think, looking back for earlier occurrences is not too heavy a burden.

`\countword` is defined only when ednotes.sty has been loaded with the option `countoccurrences`.

4 Packages related to ednotes

4.1 Overview

So far we have mentioned the packages ednotes, manyfoot, lineno, and longtable. Their names appear in figure 9, among others. Indeed, all the "strings" in figure 9 refer to package files, with the extension '.sty' omitted. We now explain those packages which have not yet been introduced.

The arrangement, in figure 9, of the package names and of the boxes in which they reside alludes to how they relate with or even "build" on each other. I.e., if the box containing the name of one package ⟨*file1*⟩ partially "covers" ("rests" on) the box containing the name of another package ⟨*file2*⟩, this means something from the following:

- ⟨*file1*⟩ does not *work* when ⟨*file2*⟩ has not been loaded (earlier), or, at least, some *option* of ⟨*file1*⟩ needs ⟨*file2*⟩. ⟨*file1*⟩ may load ⟨*file2*⟩ automatically (in one case at least this does not happen to avoid an option clash).
- ⟨*file1*⟩ *extends* or at least *modifies* the functionality of ⟨*file2*⟩.

If neither of ⟨*file1*⟩ and ⟨*file2*⟩ "covers" the other, they can be used independently from each other. —

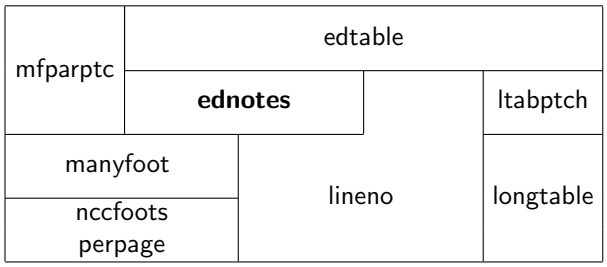


Figure 9: Packages related to ednotes

We will explain these interrelations more precisely below. First, we introduce the packages (or expand on them). We start at the bottom of figure 9 and work our way upwards.

Unless indicated otherwise, the packages are available from the CTAN directory `macros/latex/contrib/ednotes`. However, that it is available from a certain folder may mean that it is there "in disguise" only, requiring you to run other commands to actually create the `.sty` file.

4.2 Packages from other authors

The following packages have not been written by us (i.e., by U. L. or Christian Tapp; or at least "not originally").

perpage by David Kastrup is available from CTAN in `macros/latex/contrib/misc`. It switches to pagewise numbering of footnotes. This is of limited use for critical editions where footnotes usually are not numbered anyway. However, there may be commentary or introductory passages by the editor(s) between edited texts, and these may have ordinary numbered footnotes. So ednotes.sty accepts a perpage option and passes it to manyfoot.sty which in turn loads perpage.sty, with customizations.

manyfoot and **nccfoots** by Alexander I. Rozhenko are available (disguised) from CTAN in `macros/latex/contrib/ncctools`; they provide multiple "layers" of footnotes as ednotes needs them (section 3.2). These are "disguised" because what is on CTAN is actually manyfoot.dtx, nccfoots.dtx, and ncctools.ins, for extracting the .sty files on your own.

lineno by Stephan I. Böttcher (and recently extended and modified, for supporting ednotes better, by U. L., on his kind invitation) is available from CTAN in `macros/latex/contrib/lineno`; it provides numbering of lines and referring to line numbers as is needed by ednotes (section 3.1).

longtable by David Carlisle is part of the LaTeX distribution; it provides a multi-page tabular environment which lineno, or ednotes through lineno, modifies on request (`longtable` option) to enable themselves to work within (section 3.9).

4.3 Our packages not needing ednotes.sty

mfparptc: The mechanisms for typesetting footnotes as "*block formatted*" (one paragraph per page) known to us derive from Donald Knuth's suggestions in *The TeXbook* (Knuth, 1996, pp. 395 – 400). The (L)EDMAC documentation (section 'Paragraphed footnotes') and a *TUGboat* article by Michael Downes (Downes, 1990) describe shortcomings of *The TeXbook* macros, and EDMAC modifies these macros to remove those shortcomings.

Alexander Rozhenko's manyfoot does not consider these shortcomings and changes. Therefore, we wrote mfparptc to render the manyfoot block formatting mechanism working closer to EDMAC's.[1] ednotes.sty loads mfparptc.sty when given the package option `edmacpara`.

ltabptch is available from CTAN at `macros/latex/contrib/ltabptch`. We are convinced that there are three spacing bugs in longtable, see LaTeX Bug Database, tools/3180 and tools/3485, and ltabptch fixes these bugs. (In essence, there is vertical space missing above a "long" table, and the interline glue below it is in general wrongly calculated.) We tried to convince David Carlisle to take these fixes into longtable; however, he convinced us that it is better to keep the bugs/features and offer a fixing package. This preserves document layout with source files written using the defective longtable. We have proposed a compromise — due to our original conviction, ednotes.sty loads ltabptch.sty given option `longtable`, whenever it is "visible" to LaTeX. An option `nolongtablepatch` enables the user to *avoid* this.

edtable was made as an enhancement of ednotes to cover LaTeX tabular environments (as explained in section 3.9). However, it does not really need ednotes.sty and may instead be used as a mere lineno extension. lineno.sty loads it given the option `edtable`, which indeed ednotes.sty passes. The package may eventualy vanish, the options or at least the functionality will stay.

[1] We once hoped that Alexander Rozhenko would incorporate mfparptc into his manyfoot. The main reason not to do so is that mfparptc, at present, disables manyfoot's `\SplitNote`. (We hope eventually to have the time to fix this.) There are further difficulties; e.g., `\linebreak` is modified in notes to eschew one. So the name mfparptc — meaning originally a "patch" — was somewhat arrogant, sorry.

4.4 Installation and standalone packages

This section is concerned with matters of installation and with what choices users have with regard to our packages. Installation will be interesting more for actual *users* than for *readers*.

- ednotes always requires lineno and manyfoot, and loads them automatically. manyfoot itself also requires nccfoots.
- ednotes requires longtable only when given the `longtable` option, loading it automatically in the latter case.
- ednotes requires perpage only when given the `perpage` option, loading it automatically then. (manyfoot also supports the `perpage` option.)
- The `edtable` option for ednotes enables the `edtable` environment described in section 3.9. (lineno also supports the `edtable` option.)
- ltabptch can be used as a standalone patch to the standard longtable package, to overcome the problems mentioned in section 4.3. It is loaded automatically by ednotes given the `longtable` option, unless you forbid this with the further option `nolongtablepatch`.
- mfparptc can be used as a standalone patch to the manyfoot package, but take heed of the limitations discussed in section 4.3. However, use the package option `edmacpara` to use it with ednotes, rather than loading it explicitly.

There are further interdependencies, but we hope this covers typical usage.

"Visible to (LA)TEX": Of course, nothing can be loaded unless it is "visible to (LA)TEX", that is, its files can be found by (LA)TEX. This notion is, for some users, somewhat difficult. So, a few hints:

- To be found, a package file must be in a folder which (LA)TEX searches when compiling (your main document file) `\jobname.tex`.
- You may put the file in the same folder as `\jobname.tex` itself. This is inconvenient if you want to use the package for several documents, in different folders.
- You may put the file in the `contrib` folder of the main `latex` folder.
- Or, you may put it in the same folder as another .sty file that you are already using. If you have used ednotes before, just put new package files into the same folder where ednotes.sty is.
- You may find further hints at

  `tug.ctan.org/installationadvice`

  and at

  `www.tex.ac.uk/cgi-bin/texfaq2html?label=wherefiles`

Package options: lineno.sty has a lot of package options; ednotes.sty accepts them all, and merely passes them on to lineno.sty (renaming one of them; we have described four of them). We have described 13 additional ednotes.sty options (two are passed to manyfoot.sty; and there are some "obsolete" ones).

This may stimulate worrying about how to *enter* all the options that one would like to use — they may not fit into one line. Fortunately, you can safely break code lines after the commas separating the option names in the `\usepackage` command:

```
\usepackage[⟨option1⟩,⟨option2⟩,
            ...]{ednotes}
```

5 Acknowledgments

We (U. L.) are indebted to Karl Berry, David Kastrup, Jerónimo Leal, Christian Tapp, and Peter R. Wilson for having read carefully earlier drafts of this article and for all their important hints, suggestions, judgments, and corrections. Thanks also to Karl Berry for (*i*) the invitation to write this article for *TUGboat* — among our profits is that our ednotes gets a printable description for the first time — and for (*ii*) his patience and generosity with regards to all our questions on setting up the article properly, and editing our English.

We are, maybe, most indebted to Alexander I. Rozhenko and Stephan I. Böttcher for changes to their packages manyfoot and lineno in time for this article. These changes simplified the structure of our former bundle very much and, thus, simplified this article. Indeed, Stephan Böttcher passed maintenance of his lineno to me so I could make the changes in favour of co-operation with ednotes on my own — special thanks!

The ednotes *package* profited very much from intense e-mail discussions with ("rival") package authors Dominik Wujastyk, John Lavagnino, Stephan I. Böttcher, Alexander I. Rozhenko, and Peter R. Wilson as well as from substantial "complaints" by our test users Robert Alessi, Florian Kragl, and Sergei Mariev. Thanks moreover to all the people mentioned for their encouragement of our work.

Our work on ednotes developed in the course of a research project entitled 'Geschichte der Ordinalzahlanalyse und ihre Implikationen für die Philosophie der Mathematik', supported by the Deutsche Forschungsgemeinschaft (DFG) 2001 – 2004.

References

Burt, J. "Typesetting critical editions of poetry". *TUGboat* **22**, 353 – 361, 2001.

Downes, M. "Line breaking in \unhboxed text". *TUGboat* **11**, 605 – 612, 1990.

Kastrup, D. "The bigfoot bundle for critical editions". In *Preprints for the 2004 Annual Meeting*. TEX Users Group, 2004.

Knuth, D. E. *The TEXbook*. Addison-Wesley, Reading, Mass., 1996.

Lavagnino, J. and D. Wujastyk. "An overview of EDMAC: a PLAIN TEX format for critical editions". *TUGboat* **11**, 623 – 643, 1990.

Lavagnino, J. and D. Wujastyk. *Critical Edition Typesetting: The EDMAC format for PLAIN TEX*. TEX Users Group and UK TEX Users Group, San Francisco and Birmingham, 1996.

◇ Uwe Lück
Seminar für Philosophie, Logik und Wissenschaftstheorie
Philosophie-Department
Universität München
Geschwister-Scholl-Platz 1
D-80539 München
Germany
ednotes.sty@web.de

Software & Tools

Hyphenation patterns for minority languages

Kevin P. Scannell

Abstract

We present some techniques used in developing hyphenation patterns for the Irish language that we hope will be applicable to other languages with limited computational resources.

1 Introduction

Irish is one of six languages in the Celtic branch of the Indo-European family (the others are Scottish Gaelic, Manx Gaelic, Welsh, Cornish, and Breton). Typesetting enthusiasts might be familiar with the so-called "Gaelic" fonts (in Irish, *seanchló*, literally "old type") used to print the language until the early part of the 20th century, and which trace their roots back to the exquisite illuminated manuscripts produced by Irish monks in the centuries preceding the Norman conquest [4, 5, 6]:

Aoiḃinn beaṫa an ṡcoláiṗe
ḃíoṙ aṡ déanamh a léiṡinn;
iṙ ḟollaṙ doíḃ, a ḋaoine,
ṡuṗab dó iṙ aoiḃne in Éiṗinn. [1]

Once spoken by several million people, there are now perhaps only 50,000 native speakers, mostly in remote regions of the west of Ireland.[2] English remains a constant presence throughout, especially in the contexts of technology and computing. This has had the unfortunate tendency to reinforce the view, especially prevalent among the young, that Irish is "irrelevant" for modern life.

Since 1999 the author has been engaged in the development of Irish language software as a way of helping to stem the tide of language shift. Already completed are a general purpose web crawler, spell checker, and grammar checker, with a monolingual thesaurus and various localization projects currently in progress.[3] Recently, a set of TEX hyphenation patterns for Irish was completed as part of this work.[4] This topic forms the focus of this paper.

In most languages, the choice of font has no effect on hyphenation, but typesetting Irish in Romanized script (as is standard nowadays) implies some important *orthographic* changes, most notably the use of the letter 'h' to indicate "lenition", or softening, of the preceding consonant. This is indicated by a *ponc* (dot) over the lenited consonant in Gaelic type, as can be seen in the excerpt above. The current version of the patterns is designed to work with the modern orthography, but if desired can be modified to work for *seanchló* as well.

In the following sections some of the techniques used in developing the patterns will be described in the hope that they will be applicable to other minority languages. Indeed the main goal in writing this article is to encourage further work on hyphenation and other natural language processing (NLP) tools for marginalized and under-resourced languages. Some relevant statistics are presented in the final section for further reflection.

The so-called "information bottleneck" of NLP is especially acute for minority languages. It is accompanied in most cases by a lack of skilled

[1] "Lovely the life of the scholar, diligently working; you know well, good people, his is the sweetest lot in Ireland." This Gaelic font, produced using METAFONT by Ivan Derzhanski, is called eiad and is available from CTAN.

[2] The language has the nominal support of the Irish government and is taught in the schools, so a somewhat larger number of people claim fluency.

[3] http://borel.slu.edu/gaeilge.html

[4] http://borel.slu.edu/fleiscin/

software engineers, linguists, or both. This induces one to exert effort in those areas where the materials produced can be deployed as widely as possible. In this context, it is worth noting (even to an audience of TEX devotees) that Liang's hyphenation algorithm has come into much wider use over the last two or three years. The TEX hyphenation files themselves can be used directly by GNU Troff,[5] and slightly modified versions are now used by the free software packages OpenOffice[6] and Scribus.[7] There are, in addition, at least two implementations of XSL-FO processors (these convert XML data plus style sheet information into PDF and other formats) that employ TEX hyphenation (including Apache's FOP[8]).

2 General techniques

Much of what is said here is well known; in particular, bootstrapping a database of hyphenated words using `PATGEN` is a well-established technique; see, e.g. [8]. Readers interested in creating new patterns are encouraged to read Petr Sojka's papers (especially [8] and [9] with Pavel Ševeček), Yannis Haralambous' `PATGEN` tutorial [2], and the Master's thesis of David Antoš [1].

2.1 Parallel development

Just five years ago, there were virtually no Irish language lexical resources in machine-readable form. Based on the author's experience, developing a full suite of resources in parallel is easier than attempting each individually. In short, what is advocated here is a *synergistic approach* to the development of multiple NLP tools that exploits "feedback loops" and synergies between them.

As a simple illustration, consider the simultaneous development of a web crawler, text corpus, and spell checking database. The web crawler reported here uses the Google API[9] and words from the spell checking database to search for potential Irish language documents on the web; the spell checker and some statistical techniques are used to determine which documents (or sections thereof) are actually in the Irish language. These are added to the text corpus, and more statistical analyses (frequencies, character n-grams) are used to find reliable candidate words for the spell checker [7]. Such a system can be bootstrapped from a small word list, and for much of its life cycle requires no human intervention. Each of the three subsystems improves over time. Like many individuals working with small languages, I was led to employ unsupervised, statistically-based NLP not from any *a priori* fundamental belief in its effectiveness, but simply because it appeared the most viable method to achieve reasonable results in a limited timeframe.

2.2 Hyphenation and spell checking

In the context of hyphenation, feedback between the hyphenation patterns and the morphological and phonological data encoded in advanced spell checkers like `aspell` can be exploited.[10]

At the most basic level, a spell checker is really just a word list stored in some kind of hash table for efficient lookup. The standard UNIX spell checkers also offer *affix compression*. This is a way of encoding portions of the morphology of a language in an "affix file"; then, instead of storing all variants of a given word in the word list, the approach is simply to store something like a dictionary headword and a flag indicating the rules that govern inflection of the word. For heavily inflected languages like Irish, this compresses the hash table by around 70%. Below is shown a tiny chunk of the Irish affix file, showing three future endings of one kind of verb. The left hand side gives the ending to which the rule applies (as a regular expression in general) and the right hand side indicates which letters to strip off and which to add:

```
A Í M   >   -AÍM,ÓIDH
A Í M   >   -AÍM,ÓIMID
A Í M   >   -AÍM,ÓFAR
```

Since Irish is generally hyphenated according to morphological rules, the spell checker offers a powerful means to insert all of these hyphen points into the database at one go. For this, a "fake" affix file was created containing all the same rules, but which permitted an additional non-alphabetic character to appear on either side of a rule (here the use of '!' is adopted since Irish has quite a few explicitly hyphenated words). A command line option to the spell checker allows one to expand all affix flags from the (unmodified!) word list according to these rules, resulting in a rich initial set of hyphen points. One bootstrapping iteration with `PATGEN` handled all irregular verbs which weren't encoded in the affix file.

To convince the reader of the importance of parallel development, this subsection is closed with an example of feedback from the hyphenation patterns

[5] `http://www.gnu.org/software/groff/groff.html`
[6] `http://www.openoffice.org/`
[7] `http://web2.altmuehlnet.de/fschmid/`
[8] `http://xml.apache.org/fop/index.html`
[9] `http://www.google.com/apis/`
[10] `http://aspell.net/`.

back to the spell checker. The easiest example of this is the input to the "metaphone algorithm" implemented as part of `aspell`. This algorithm depends on the existence of a "coarse" phonetic encoding of your language that can be used to improve suggestions when a misspelling is encountered.[11] Having an accurate hyphenation database in place before attempting such an encoding offers a significant efficiency. For instance, the words *garbhuille* (*gar* + *bhuille* lit. "near + stroke" — an approach shot in golf) and *garbhghlórach* (*garbh* + *ghlór* + *ach* lit. "rough + voice + ish" — raucous) share their first five letters, but these are pronounced quite differently in each case. Having the hyphenations in place while constructing the phonetic rules allows one to avoid numerous special cases dealing with situations like this.

2.3 Print sources and human informants

Another serious problem worth mentioning, as it surely faces most other minority languages, is the lack of explicit standards for, or printed dictionaries of, Irish hyphenations. The only general observation beyond debate is that Irish is best hyphenated according to etymological and morphological rules. Knowing this, it becomes quickly and painfully apparent when a given text has been hyphenated according to an English (syllabic) computer algorithm, or, as apparently happened quite often during the early Irish revival, by a monolingual English-speaking compositor.

The most abominable examples of this result from the the convention, noted above, of using an 'h' in Roman type to indicate lenition; this rule has the corollary that one should never split the 'h' from the preceding consonant. Unfortunately, examples like *com-halta* or *bót-har* can be found in abundance in printed books. Matters are somewhat complicated by the fact that 'h' appears occasionally in loanwords and in such contexts often *is* a good hyphenation point: *Bói-héam-ach* ("Bohemian").

Nevertheless, the author was able to assemble enough suitable printed material to populate the initial hyphenation database manually.[12] Originally it was hoped to extract, automatically, hyphenated words from the many online PDF documents produced by the Irish government, but (presumably because of the lack of proper hyphenation technology) many of these are set with a ragged right!

This work also benefited greatly from the input of many Irish speakers who checked over the hyphenations produced by early versions of the patterns on the top 1000 most frequent Irish words; see `http://borel.slu.edu/fleiscin/mile.html`.

2.4 `PATGEN` esoterica, final results

One of the most difficult aspects of using `PATGEN` effectively is the choice of correct parameters. Some good heuristics are offered in [1], with actual examples (size-optimized, precision-optimized, etc.) in [9]. I found that with (the usual) five levels of hyphenation, I consistently ended up with a couple hundred bad hyphenations; adding a sixth (inhibiting) level with parameters $(1, 1000, 1)$ disposed of these and only added 1K or so to the final pattern file. Here is the full set of parameters which worked best:

Level	Lengths	Parameters
1	2 . . . 4	(1,2,30)
2	2 . . . 5	(1,2,30)
3	3 . . . 6	(1,2,6)
4	3 . . . 7	(1,2,6)
5	3 . . . 8	(1,1000,1)
6	3 . . . 9	(1,1000,1)

The result is a large set of patterns (about 6000) but an extremely accurate one: no bad hyphens and just 10 missed hyphens from a database of 314,639 possible hyphen points in 234,789 words.

3 Other languages

According to the Ethnologue database (`http://www.ethnologue.com/`), there are more than 6800 living languages; at least 2000 have some form of writing system (based on a count of the number of languages with at least partial Bible translations).[13] By one rough count, however, there are only 36 languages having a reasonably complete desktop computing environment available.[14] Only half of the world's population are native speakers of one of these 36 languages, meaning some three billion people have no way of using a computer in a native language context (ignoring the more fundamental problems of poverty and illiteracy for many of these same three billion).

[11] This is especially important for Irish, which has many silent consonants and which underwent a major spelling reform in the 1950's. For example, the top suggestion made by `aspell` for the pre-standard form *imfhiosach* is, correctly, *iomasach*, which has the same phonetic encoding.

[12] For the Irish speakers among the readership, the best choices were books published by Sáirséal agus Dill during the 1950's and 1960's.

[13] For a nice discussion of this question, see `http://www.ogmios.org/117.htm`.

[14] For instance, version 3.1 of the KDE desktop for GNU/Linux is at least half translated for exactly 36 languages; Windows and Mac localizations make up a (small) proper subset of these.

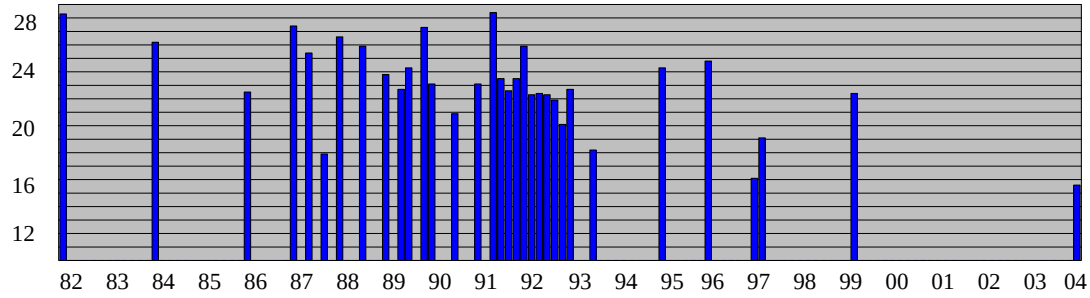


Figure 1: TEX hyphenation patterns by initial release date

The numbers are similar for TEX hyphenation patterns; using the table in [9], the CTAN archives, and some web searching, I found at least some mention of the existence of patterns for 34 natural languages (not surprisingly, 28 of these appear on the list of 36 above). It seems, though, that production is slowing down (although other explanations are possible, e.g. lack of publicly released materials). In figure 1, the horizontal axis represents time (labeled with two-digit years). Each bar indicates the initial release of TEX hyphenation patterns for a new language, with the height of the bar given by the base 2 logarithm of the number of native speakers. For example, Estonian (patterns released in 1992) has a hair over 2^{20} or one megaspeaker. The far left represents Liang's thesis [3] (assuming $2^{28.3} = 340,000,000$ native English speakers) while the far right represents the patterns for Irish (assuming $2^{15.6} = 50,000$ native speakers). Note that in the past decade, patterns for just six new languages have been released (Romanian, Indonesian, Sorbian, Basque, Mongolian, and Irish) with just one in the past five years.

There is a small research community concerned specifically with NLP for minority languages (see, for instance, `http://193.2.100.60/SALTMIL/`), but their work is confined largely to the European sphere, encompassing perhaps thirty languages beyond the three dozen or so noted above. The author has undergraduate students currently applying some of the techniques described in this paper to the Maori and Inuktitut languages, but this, of course, is just a drop in the bucket.

It is worth emphasizing, in closing, the importance of an open source approach to these problems, leveraging the collective effort of small communities, and transcending the purely market-driven approach that has led to the current dismal state of affairs. It is hoped that the work reported in this paper makes a small contribution to addressing this situation.

References

[1] David Antoš. Generation of Patterns with the `OPatGen` Program. `http://www.fi.muni.cz/~xantos/patlib/thesis.html`, 2001.

[2] Yannis Haralambous. A small tutorial on the multilingual features of PatGen2. `http://www.ctan.org/tex-archive/info/patgen2.tutorial`.

[3] Franklin M. Liang. *Word Hy-phen-a-tion by com-puter*. Ph.D. thesis, Stanford University, 1983.

[4] E. W. Lynam. *The Irish Character in Print*. Barnes and Noble Inc., New York, 1969.

[5] Dermot McGuinne. *Irish Type Design*. Irish Academic Press, Baile Átha Cliath, 1992.

[6] Timothy O'Neill. *The Irish Hand*. The Dolmen Press, Port Laoise, 1984.

[7] K. P. Scannell. Automatic thesaurus generation for minority languages: An Irish example. In *Actes de la 10e conférence TALN à Batz-sur-Mer*, volume 2, pages 203–212. ATALA, 2003.

[8] Petr Sojka. Hyphenation on Demand. *TUGboat*, 20(3):241–247, 1999.

[9] Petr Sojka and Pavel Ševeček. Hyphenation in TEX — Quo Vadis? *TUGboat*, 16(3):280–289, 1995.

◇ Kevin P. Scannell
Department of Mathematics and Computer Science
Saint Louis University
St. Louis, MO 63017
USA
`scannell@slu.edu`
`http://borel.slu.edu/`

(LA)TEX, genealogy, and the LifeLines software

Andrew Caird

Abstract

Another example of (LA)TEX's utility as a portable, platform-independent typesetting system is in its use as an output format for genealogical information from the freely-available cross-platform genealogy software LifeLines.

Introduction

Genealogy, the study of family histories, is a hobby that can involve many generations and produce very large amounts of data. For this information to be useful from generation to generation (or even from year to year), it must be stored in a format that is portable between computer systems, useful even without a computer, and flexible enough to produce histories from different perspectives.

While (LA)TEX clearly meets the requirements of consistency and portability, it is, just as clearly, not a genealogical system. However, the same traits that make (LA)TEX so good at long-term information representation are needed in a genealogical package. One such package, available under the permissive open source MIT License [1], is called LifeLines.

LifeLines: A brief introduction

This is not a LifeLines journal, or even a genealogy journal, so this introduction to LifeLines will be brief. Please see the LifeLines home page [2] for more information.

LifeLines is a console-based program for Unix, Windows, and Macintosh that manages genealogical relationships using a GEDCOM (GEnealogy Data COMmunications) [3] database. GEDCOM is a very prevalent format for genealogical data; it is maintained by The Church of Jesus Christ of the Latter Day Saints. Because LifeLines is freely available and works on many platforms, storing data that has such a potentially long lifetime in this software is much less worrisome than storing such data in a proprietary piece of commercial software that runs on only one platform. Additionally, LifeLines can import and export GEDCOM data to and from other programs.

Installing LifeLines LifeLines can be installed on Unix, Windows, and Macintosh OS X. For instructions on installing LifeLines on Windows, Mac OS X, or specific Unix variants please see the aforementioned LifeLines web site. There are compiled packages for Windows, Linux (RPM), and Mac OS X that make installation straightforward. For the general case, however, following are the basic steps to install LifeLines on a Unix system with the GNU C compiler available. First download the latest version of LifeLines from the web site mentioned above; this will be a file named similar to `lifelines-3.0.29.tar.gz`, depending on the version. Uncompress the file by typing `gunzip lifelines-3.0.29.tar.gz` and extract the tar file by typing `tar xf lifelines-3.0.29.tar`. Now change directories into the source directory, in this case that is `lifelines-3.0.29`, and compile the software with `./configure` and then `make`. On a 266 MHz Pentium II this takes less than 10 minutes.

After compiling LifeLines, install it by typing `make install` in the source directory. Depending on where you are installing LifeLines, this step may need to be performed as a privileged user (typically `root`). Installation options can be specified when you run the `configure` script.

For more information on installation and build options, run `./configure --help`, read the `README` and `INSTALL` files, or ask for help on the `LINES-L` mailing list [2].

Confirm that there is a `reports` directory where you installed LifeLines and that it contains files that end in `.ll`. If you don't see the directory, you'll need to create one by hand and tell LifeLines to look there for its report programs. To do this, choose a sensible location (a good choice is the `share` directory near where you installed LifeLines; that is, if you installed LifeLines in `/usr/local`, you would then put the report programs in `/usr/local/share/reports`) and create a directory called `reports`.

In the LifeLines source distribution you'll find all of the report programs in the `reports` directory. Copy those files to the directory you just created. The last step is to tell the LifeLines program where to find the report programs. When LifeLines starts, it looks for configuration information in a file called `.linesrc` in your home directory. A sample `.linesrc` is distributed with the LifeLines source. Copy the sample configuration file to your home directory and modify the line that starts with `LLPROGRAMS` in that file to reflect the location of your new `reports` directory.

If you haven't seen any errors during any of these steps, then you have successfully installed LifeLines on your system. If you did see errors, there is an active LifeLines discussion list called `LINES-L` that may be able to help. More information on this is available at the LifeLines web site [2].

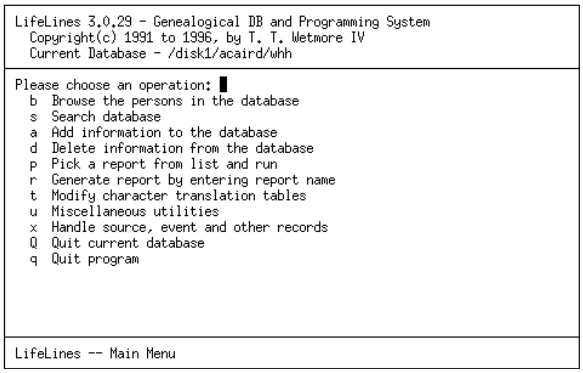


Figure 1: The main LifeLines screen

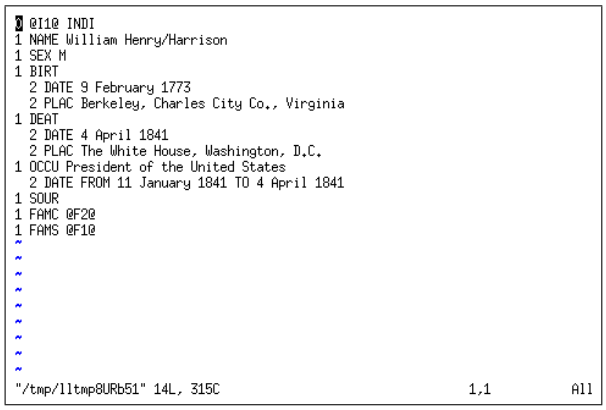


Figure 2: Editing GEDCOM data with `vi`

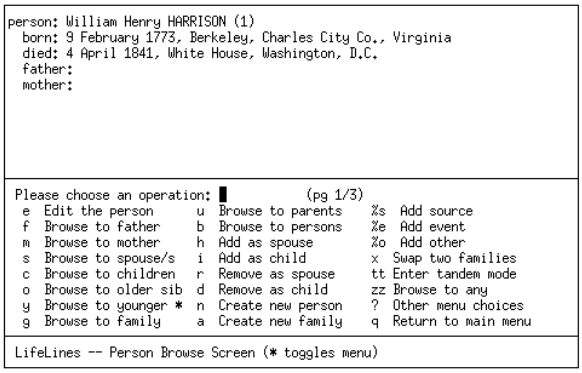


Figure 3: Viewing a person's data in LifeLines

Using LifeLines To use LifeLines you enter data in the GEDCOM format and establish relationships between people. Much like LaTeX, LifeLines focuses on the information, not the presentation; in fact, the LifeLines interface is notably non-flashy (although a richer GUI is under development), but is fast and efficient. Typical LifeLines screens on a Unix system are shown in Figures 1 and 3.

To demonstrate the basic usage of LifeLines, we'll step through the creation of a family consisting of the ninth president of the United States, William Henry Harrison, his wife, children, and parents.

LifeLines installs into `/usr/local/bin` by default on Unix systems, which is typically already in the search path. If you have installed LifeLines elsewhere, you will need to either specify the full path to LifeLines to start it, or put that directory in your `PATH` environment variable. For the purposes of this example, I will assume that LifeLines is installed in the default location, and that that location is included in your `PATH` environment variable.

Before starting LifeLines, you should create a directory where it will store your genealogical databases; since we are storing information about William Henry Harrison, I've created a directory called `whh`. To start LifeLines type `llines`. The first question it asks is which directory holds the LifeLines database. In my case, I enter `/disk1/acaird/whh` and am then presented with the main LifeLines screen, shown in Figure 1. At this point, you have an empty database, and need to add a person. We'll start by adding the President; choose `a` from the main menu, then choose `p` to add a new person from the next menu. This will start your editor or default to the `vi` editor. (Setting the `EDITOR` environment variable specifies the application called to edit LifeLines entries.) The editor brings up a blank template with places for the basic data on name, sex, birth, and death. You can add other fields in this file for other information; see the LifeLines manual and the GEDCOM standard [3] for allowed fields. A populated template for William Henry Harrison, with the additional occupation field (`OCCU`) added, is shown in Figure 2. After editing the template, save the (temporary) file and quit the editor. LifeLines will then confirm that you want to accept this information; if you have made mistakes while editing the information in the record and forgot what you changed, you can say no here and the record is unchanged. LifeLines will display the person in its browse mode; this is shown in Figure 3. It doesn't show all of the information about the person, but you can still see it if you edit the person's record, and the extra information will be included in certain types of reports.

In the menu, as shown in Figure 3, there are options in the middle column to "Add as spouse", "Add as child", and "Create new family". Using these options and the "Add information to the database" option from the main menu, you can construct families and generations.

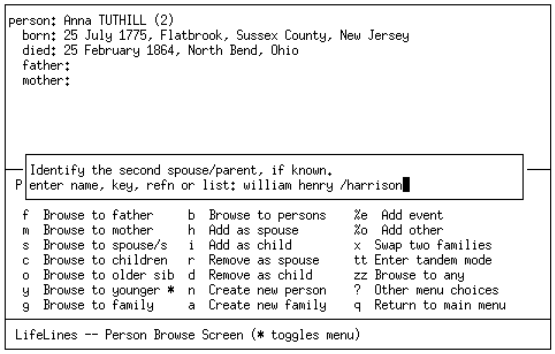


Figure 4: Adding a spouse to create a family in LifeLines

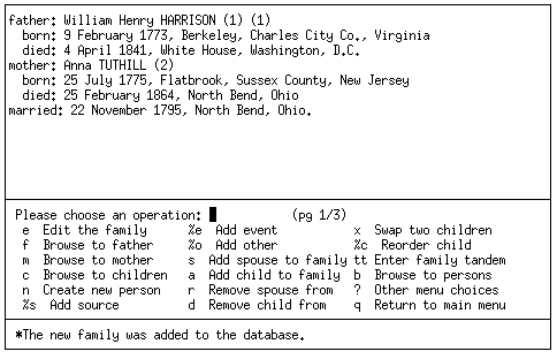


Figure 5: LifeLines's display of a family

Continuing on in our genealogy of President Harrison, we add his wife, Anna Tuthill, by returning to the main menu, selecting "Add information to the database" and adding her information. When, after editing the template, LifeLines returns you to the display of Anna Tuthill's information in browse mode, choose option `a`, "Create new family". From the next menu, choose option 2 to tell LifeLines to use Ms. Tuthill as one of the spouses. When

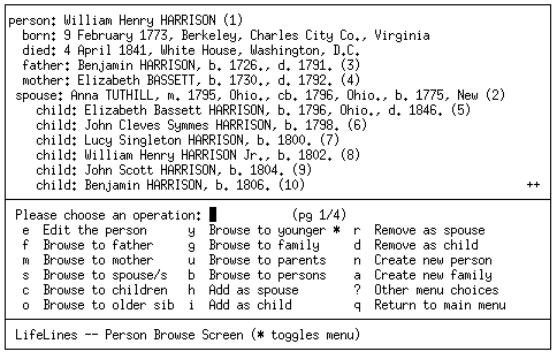


Figure 6: LifeLines's display of William Henry Harrison's complete record

prompted for the other spouse, type William Harrison's name with a forward slash (/) before his last name as shown in Figure 4. (LifeLines always expects a slash before the last name when it asks for a person to search for.) After creating this family, LifeLines will return you to browse mode, focused on this family; this "family-centric" view of the data is shown in Figure 5. By using the "Browse to father" and "Browse to mother" options, you can get to views of the data focused on individuals instead of families.

At this point, you simply continue to add people and associate them with others via wife–husband relationships, or parent–child relationships. William Henry Harrison's complete record including his spouse, parents, and children is shown in Figure 6.

LifeLines's LaTeX output

In addition to the already stated advantages of LifeLines (availability, portability, and based on non-proprietary standards), one of the most powerful features of LifeLines is that its output is all based on external "programs" written in the LifeLines language; these programs end in the `.ll` extension by convention, although LifeLines will read these programs regardless of the extension on the filename.

Included with LifeLines are more than 80 programs to generate reports based on the data entered into LifeLines' database. There are programs to create the traditional family-tree or a circular family-tree in PostScript, to create HTML files of calendars with all of the events in a family history in listed in them, to create a report of all of the surnames in the database with their corresponding SOUNDEX codes, and, most interestingly for our purposes, a report program to create a book in LaTeX with many of the details of a family history.

Because (LA)TeX lends itself so well to automated document creation, produces beautiful output, and has tools for easy maintenance of tables of contents, indices, and can include graphics, it is a natural choice for a program like LifeLines for cases where the output is complex.

The report program that creates a LaTeX book is called `book-latex.ll`. This program reads the data from LifeLines and produces a LaTeX file based on the book class that is made up of sections for each generation of a family and chapters for each family. Within each section is a paragraph (or several paragraphs) about each family member in that generation. These paragraphs are more or less rich depending on the information available. In addition to plain text, you can include graphics with careful use of LifeLines' and GEDCOM's `NOTE` field and the `\image`

command in `book-latex.ll`. `book-latex.ll` also uses LaTeX's integration with `makeindex` to create an index by surname of everyone in the book. The book can be created from a descendant or ancestor perspective, and any person or people in the LifeLines database can be the "root" of the book. For example, I can create a book with both my wife and me at the "root", so the details of all of our direct ancestors are in the book. However, for my sister-in-law, I can create a book with her at the "root" and it will contain very little of my family (I would only be mentioned in that I married her sister), and the rest of the book is about her family.

Running the `book-latex.ll` program After assembling the genealogical information, entering it into LifeLines, scanning photos and documents such as birth, marriage, and death certificates, it is a relatively simple matter to create a book with a very useful table of contents and index that is a complete family history, using the `book-latex.ll` LifeLines program.

To run `book-latex.ll`, to the main menu of LifeLines and choose option "r". When prompted for a program to run, type `book-latex`. (If LifeLines gives you an error here, go back to the installation section and confirm that you have the report programs installed and the `.linesrc` file set up correctly.)

At this point, `book-latex.ll` will prompt you for its many options. The first choice is whether you want and ancestor or descendant book; I think the ancestor book is more interesting, so for this example, choose that option. The next choice is what level of notes you want included in the book. GEDCOM has a `NOTE` field that you can include if you want in the report. In this case we don't have any notes, so choose the first option, `TEXT`. `book-latex.ll` next asks you to identify the person you want to include in your book. Type the name in the standard Firstname/Lastname LifeLines format and confirm your selection. `book-latex.ll` will then ask you again to choose a person; if you have a second person you'd like to include in the book (most commonly a spouse, to make a complete family tree) enter that person's name, or, if you don't want to include a second person simply press enter. `book-latex.ll`'s next question is the name of the output file; choose something ending in `.tex`. You will then be prompted for the name of the author of the book and the place (city, state, country) for the copyright; enter appropriate values or press enter to leave these blank. The final question `book-latex.ll` asks is for the name of a file to include as an introduction. This is a standard LaTeX file without the preamble that is included in the beginning of the report. After you answer the last question, the report program will run and you'll see the list of the people included in the report on the screen.

You now have a LaTeX file that is the report (if you didn't specify a path, it is in the same directory as your LifeLines databases). The LifeLines distribution includes the file `tree.tex` which is needed to process the report. You can add this file to your TeX installation or simply put it in the same directory as the LifeLines LaTeX output file.

To turn the LifeLines-generated LaTeX file into a DVI file that you can print or convert to PDF, you need to run LaTeX and `makeindex` on it several times, as follows:

1. Run LaTeX on the generated file. If this is the first time you are doing this LaTeX will stop with the error:
   `! LaTeX Error: File './whh.ind' not found.`
   and prompt you for the name of the index file. Since you can't have one until running LaTeX at least once, simply press Enter and let LaTeX continue on. In the example I've been using up to now, I named my LaTeX file `whh.tex`, so the first command is: `latex whh`
2. Next run `makeindex` to create an index file that LaTeX can read. In our example, the command is simply: `makeindex whh`
3. Run LaTeX again on your output file. This time it won't complain about any missing files but it is likely to warn you that labels may have changed and you should run LaTeX one more time. Do so one final time, and the resulting DVI file will contain a nicely formatted and indexed report. The commands are simply two more invocations of: `latex whh`

Figure 7 shows an example of LifeLines's LaTeX output. The output is easily modified by editing the `book-latex.ll` script that LifeLines uses. If you are comfortable with basic programming and LaTeX concepts, you will find `book-latex.ll` easy to understand and modify. You may want to make modifications because you may notice that the output of LifeLines has some peculiarities: years without months are preceded with an extra comma; spaces after periods are post-sentence spaces. Hand-editing before publication may be necessary. Alternatively, the LifeLines maintainers would, I'm sure, be happy to accept modifications to `book-latex.ll` to address these oddities.

1 William Henry HARRISON

William Henry Harrison, the son of Benjamin Harrison(**2**) and Elizabeth Bassett(**3**), was born on February 9, 1773 in Berkeley, Charles City Co., Virginia. He married Anna Tuthill on November 22, 1795 in North Bend, Ohio. Anna was born on July 25, 1775 in Flatbrook, Sussex County, New Jersey and died on February 25, 1864 in North Bend. William died on April 4, 1841 in The White House, Washington, D.C. and was a President of the United States from 11 January 1841 to 4 April 1841.

Children of William Henry Harrison and Anna Tuthill:

i. Elizabeth Bassett Harrison was born on September 29, 1796 in Fort Washington (now Cincinnati), Ohio. She married John Cleves Short on June 29, 1814. John was born in , 1792 in USA and died in , 1864 in USA. Elizabeth died on September 27, 1846.

ii. John Cleves Symmes Harrison was born on October 28, 1798.

iii. Lucy Singleton Harrison was born on September 5, 1800.

iv. William Henry Harrison Jr. was born on September 3, 1802.

v. John Scott Harrison was born on October 4, 1804.

vi. Benjamin Harrison was born on September 8, 1806.

vii. Mary Symmes Harrison was born on January 22, 1809.

viii. Carter Bassett Harrison was born on October 26, 1811.

ix. Anna Tuthill Harrison was born on October 28, 1813.

x. James Findlay Harrison was born on June 21, 1818.

Figure 7: A segment of typical `book-latex.ll` output

Conclusion

While (LA)TEX is excellent for typesetting "human-created" documents, it is also very good at typesetting computer-generated documents. LifeLines takes advantage of this, along with LATEX's strong cross-platform support, to produce well formatted and indexed genealogical reports. With some experimenting and relevant names and dates, you will quickly find using LifeLines and its reports an interesting and addictive pastime, with the added benefit of creating something you can share with the rest of your family and pass on to the next generations, secure in the knowledge that the information about your ancestors and contemporaries will survive operating system upgrades, hardware changes, and all of the other vagaries of modern computer use.

Thanks

Thanks to the original author of LifeLines, Thomas T. Wetmore IV; the current SourceForge project maintainers, Marc Nozell and Perry; the author of the `book-latex.ll` script, Dennis Nicklaus; and all of the current developers.

References

[1] Information on the MIT Open Source Software License: `http://www.opensource.org/licenses/mit-license.php`

[2] The LifeLines web site: `http://lifelines.sourceforge.net/`

[3] The GEDCOM web site: `http://www.gendex.com/gedcom55/55gctoc.htm`

◇ Andrew Caird
1065 Chestnut St.
Ann Arbor, MI USA
`a_caird@yahoo.com`

Generating LaTeX documents through Matlab

S. E. Talole and S. B. Phadke

Abstract

Matlab, along with its family of toolboxes, is widely used software for analysis and design of a large number of real life engineering problems encompassing areas such as signal processing, control system design and so on. The output of a problem solved using Matlab can be included in a Microsoft Word document by using the *Notebook* supplied with Matlab or the *Matlab/Simulink Report Generator Toolbox* available separately. In academic institutions, documentation is commonly done using LaTeX. While the graphics generated by Matlab can be saved in PostScript form and then included in a LaTeX document, at present there is no way to directly include numerical data and text in a LaTeX program. Manual inclusion of such data is error prone and time consuming.

The objective of this paper is to present the idea of generating notes, precis or parts of books through the use of Matlab engine for writing LaTeX programs. When sets of data or graphs are to be included in a LaTeX document, the programming power of Matlab can be effectively employed. The software presented here is written with feedback control applications in mind. The software needs the *Control Systems Toolbox* Matlab module and augments its functionality.

1 Introduction

Matlab is one of the most widely used environments for solving real-life engineering problems. It is a valuable tool in teaching and research in several disciplines, such as control engineering, signal processing and so on. A number of books have been written illustrating the power and use of Matlab for solving practical problems. While solving a problem, it is very important to document it along with its solution; normally, this is done by typesetting the problem and its solutions separately. Graphical results of the solution are incorporated by employing copy and paste technique as is done with Microsoft Word. Such a procedure is highly time consuming and prone to errors. Even when the *Notebook* supplied with Matlab is used, the output is of a poor quality, especially when the document contains a lot of mathematics.

LaTeX (Buerger, 1990) is a highly regarded and widely used publically available typesetting environment. Based on TeX, developed by Donald Knuth (Knuth, 1986), LaTeX provides a powerful means for preparing high quality typeset documents and has become a de facto standard for submitting technical papers in international journals and conferences (Kwakernak, 1996). Many academic institutions as well as universities and research establishments use LaTeX for typesetting.

In view of the wide use of LaTeX, we felt that when a problem is solved using Matlab, a documentation of the problem and its solution in LaTeX would be highly useful. The power of Matlab as a computational engine needs to be combined with the power of LaTeX as a typesetting engine to achieve this end. The purpose of this paper is to present a small development to fulfill this need. The Matlab programs described in this paper themselves generate the necessary LaTeX documents. Whenever repeated typesetting tasks are involved, the programming power of Matlab has been used to do the same.

2 Generating LaTeX through Matlab

The software described in this paper is a set of Matlab functions or script files designed to get time and frequency response and write the text and/or graphical output to a LaTeX file. The LaTeX file is then processed separately. For example, the functions `PlaceLatex` and `BodeLatex` developed here do everything that the standard functions `place` and `bode` in the *Control Systems Toolbox* do, while generating a LaTeX file as output. As far as the user is concerned, the only difference between the standard `place` and `bode` functions of Matlab and the `PlaceLatex` and `BodeLatex` functions is that the latter needs one extra argument, a string specifying a filename in which the results are stored. In addition, the legends and the plots are exported in LaTeX format without any additional manual entry.

Wherever several results of a similar nature are needed, such as graphs generated by varying one or more parameters, the programming power of Matlab itself is used to do the job. This concept is illustrated by writing a Matlab script to obtain graphs of a step response of a second order system as the damping ratio is varied from 0.1 to 1 with a step of 0.1. When this script file is executed in Matlab, it generates a LaTeX file containing all the graphs. In the present version, functions have been written to accomplish most of the common tasks needed in time and frequency response and stability analysis, and write corresponding LaTeX documents.

The LaTeX documents are generated by utilizing the file I/O and string functions provided in Matlab. The LaTeX commands provided in the *Symbolic*

Math Toolbox can facilitate the document generation but are not essential. Knowledge of basics of feedback control is utilized in generating appropriate strings by interpreting the results generated by Matlab so that the documentation will appeal to the control engineer. In the next section the use of `PlaceLatex` and `BodeLatex` for automatic generation of a LaTeX document is illustrated by examples. Further, a Matlab script file which generates a LaTeX document by employing the programming power of Matlab is also presented.

The LaTeX file generated by these programs can be processed in any standard LaTeX implementation. It may be noted that many LaTeX implementations are freely available. On the Matlab side, the m-file (Matlab scripts) functions require the *Control Systems Toolbox*. Following similar procedures, several other functions can be developed for step response, Nyquist stability, Routh stability, lead and lag compensation, etc., to provide for commonly needed tasks in control system analysis and design.

3 Examples

In this section, the use of the new `PlaceLatex` and `BodeLatex` functions, as well as the script file developed in the present software, are presented.

3.1 Pole placement problem

Consider a dynamic system, the state space model of which is

$$\dot{x} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 20.601 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -0.4905 & 0 & 0 & 0 \end{bmatrix} x + \begin{bmatrix} 0 \\ -1 \\ 0 \\ 0.5 \end{bmatrix} u \quad (1)$$

It is desired to place the closed loop poles at $-2 \pm j3.464$, -10, -10. The `K=place(a,b,p)` command available with the *Control Systems Toolbox* (Mathworks, 1998) gives the state feedback gain vector K such that the closed loop poles of the system are as specified in vector p. The arguments a and b are the system and input matrix of the open loop system. However, if one wishes to document this problem and its solution, one will have to typeset it separately by manually entering the arguments of the Matlab function as well as the output generated by Matlab. We wish to avoid this manual entry.

To generate the documentation automatically, a new Matlab function `PlaceLatex` has been written, which when invoked along with its arguments, generates a LaTeX document. For example, using

```
PlaceLatex(a,b,p,'placex.tex')
```

creates `placex.tex`, which can be processed like any LaTeX document. The `PlaceLatex` function is written by using the low level Matlab programming commands. The function commences with the function definition:

```
function out = PlaceLatex(a,b,p,filename)
```

which declares that the function needs four arguments: the Matlab input data and the LaTeX file to write, as described earlier.

Next, the software checks whether four arguments are in fact specified:

```
if nargin ~= 4
   error('Must have 4 input arguments!')
end
```

The file named by `filename` as specified by user is opened for writing:

```
fid=fopen(filename,'w');
```

The function exports text as well as numerical results through the Matlab command `fprintf`. The function `fprintf` can be used in a variety of ways. Static text relevant to the problem can be generated by commands like

```
fprintf(fid,'Consider a system the state
space model of which is \n');
```

Similarly, the statement

```
fprintf(fid,'\\begin{center}
$ \\dot x$')
```

generates simple LaTeX code for opening the centering environment and writing $\dot{x}$. Backslash is an escape character in Matlab strings, thus we double it to get one backslash in the LaTeX output file.

The most interesting use of `fprintf`, however, is for writing dynamic text that is not physically entered by the user. An example of this is

```
fprintf(fid,'%g &',a(i,j))
```

A section of code that can generate an array a of numbers is as shown below:

```
ao=length(a);
for i=1:ao
   for j=1:ao
      if j==ao
         fprintf(fid,'%g \\\\',a(i,j));
      else
         fprintf(fid,'%g &',a(i,j));
      end
   end
end
```

This Matlab code will insert the elements of a square matrix in a LaTeX document. With the addition of suitable commands, full code for displaying the matrix can be generated. A point worth noting is

that the code remains the same irrespective of the order of the matrix.

Returning to our `PlaceLatex` function, a technical check is made for the controllability of the system; this code is omitted. Then, the pole placement design is carried out with:

```
k=place(a,b,p);
```

The resulting gain matrix `k` is then written to the user's file with:

```
fprintf(fid,'\\begin{center}
\\mbox{State feedback
gain matrix,}
$ K = [') for i=1:ao
   fprintf(fid,' \\ %g',k(i));
end fprintf(fid,']. $
\\end{center}')
```

Finally, the output file is closed:

```
fclose(fid);
```

An excerpt from the typeset LaTeX documentation for the present problem follows.

Pole placement:

Consider a system the state space model of which is

$$\dot{x} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 20.601 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -0.4905 & 0 & 0 & 0 \end{bmatrix} x + \begin{bmatrix} 0 \\ -1 \\ 0 \\ 0.5 \end{bmatrix} u$$

The open loop poles of the system are

$$\text{Open loop poles} = [\ 0\ 0\ 4.53883\ \ -4.53883]$$

The open loop system is unstable as pole/s are lying in RHP. The coefficients of the open loop characteristic polynomial are :

$$\text{Open loop characteristic polynomial coefficients} = [\ 1\ \ 0\ \ -20.601\ \ 0\ \ 0].$$

The desired closed loop poles are given as : [. . .]

$$\text{State feedback gain matrix,}$$
$$K = [\ -298.146\ \ -60.6965\ \ -163.092\ \ -73.3931].$$

Two points about this output are worth noting:

1. None of the numerical data is physically entered.
2. The program for generating the document is totally independent of the problem being solved and documented.

3.2 Bode plots

For our next example, consider a feedback system the open loop transfer function of which is given as

$$G(s)H(s) = \frac{20s + 20}{s(s^3 + 7s^2 + 20s + 50)} \qquad (2)$$

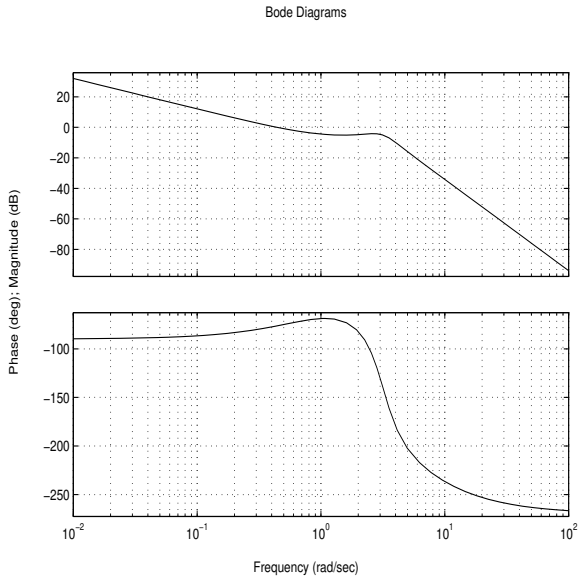


Figure 1: Open loop bode plots

To generate its bode plot, one can use the `bode` command as `bode(num,den)` where $num = [20\ 20]$ and $den = [1\ 7\ 20\ 50\ 0]$. Execution of this command in Matlab with *Control Systems Toolbox* displays the bode plot of the considered transfer function.

To generate the documentation of this problem automatically, a new Matlab function `BodeLatex` has been written, analogous to `PlaceLatex`. For example, `BodeLatex(num,den,'bodex.tex')` creates `bodex.tex`, which when *latexed* generates the following (excerpted).

Bode Plot:

This is an example of Bode plot report generated through Matlab. Consider a feedback system having open loop transfer function as

$$G(s)H(s) = \frac{20(s+1)}{s(s+5)(s^2+2s+10)}$$

[. . .]

The open loop bode plots for the considered system are as shown in Figure 1. To see the gain and phase margins, one can use the *margin* command which gives the bode plots as shown in Figure 2.

[. . .]

Since $\omega g < \omega p$, the system is stable.

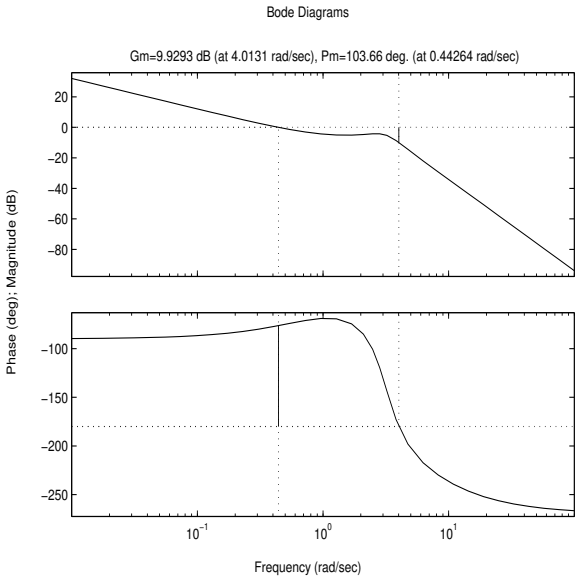


Figure 2: Open loop bode plots with gain and phase margins

3.3 Matlab script files

Now let us consider a task of plotting a step responses of a second order system

$$G(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}$$

as its damping ratio, ζ is varied from 0.1 to 1.0 in the steps of 0.1, thus, a series of 10 graphs. If one wishes to include these 10 graphs in a LaTeX document, it's clearly best not to write the code for inclusion of graphics, caption and label, and copy it (with modifications) 10 times.

It is important to note that here a task (inclusion of a graph in LaTeX document) is repeated, and so the power of Matlab programming itself can be invoked as is evident from the following portion of a Matlab script file:

```
%Plots of second order system response
%with varying damping ratio
for i=1:10, zeta=i/10;
  s=num2str(zeta);
```

This is followed by code (not shown here) that generates the step response.

Next, the code for generating distinct file names and storing the graphs as EPS files and then calling the graphs in `\includegraphics` commands with an appropriate caption including the corresponding value of ζ is shown. This code uses the Matlab function `num2str` to convert numbers to strings and `strcat` to concatenate strings.

```
s=num2str(i);
filename = strcat('sresp',s);
print('-deps', filename);
fprintf(fid,'\\begin{figure}\n
\\begin{center}\n\\includegraphics
[width=3in,height=3in]{sresp%i',i);
fprintf(fid,'.eps}\n
\normalfont\normalfont \\
\caption{Step response for $ \\
zeta=%3.1f',zeta);
```

Execution of this file in Matlab generates a file which consists of inclusion of all ten graphs of step responses of a second order system

$$G(s) = \frac{25}{s^2 + 10\zeta s + 25}$$

as ζ is varied from 0.1 to 1.

4 Conclusion

In this paper, software which generates a LaTeX document from Matlab is introduced. The software alleviates the need to typeset the problem and its solution separately by generating the documentation automatically in LaTeX. The document generation is nearly transparent to the user, i.e., the user need not know LaTeX in great detail.

It is hoped that such suite of m-files will be immensely useful to teachers and students and authors of books on control systems. In a modification of these m-files, it is intended to make the documentation entirely transparent to the user, eliminating altogether the need to know LaTeX.

References

Buerger, D. J. *LaTeX for Scientists and Engineers*. McGraw-Hill, New York, 1990.

Knuth, D. E. *The TeXbook*. Addison-Wesley, 1986.

Kwakernak, H. "Electronic Text Processing, AUTOMATICA and Elsevier". *Automatica* **32**(3), 303–304, 1996.

Mathworks. *Control Systems Toolbox User's Guide*. The MathWorks, Inc., 1998.

◇ S. E. Talole
Scientist, G.M. faculty
Institute of Armament Technology
Girinagar, PUNE-411 025
setalole@hotmail.com

◇ S. B. Phadke
Scientist, G.M. faculty
Institute of Armament Technology
Girinagar, PUNE-411 025
sbphadke@hotmail.com

MlBibTeX's Version 1.3

Jean-Michel Hufflen

Abstract

We present the features of the new version of MlBibTeX, a new multilingual implementation of BibTeX, the bibliography program associated with (LA)TeX. The main point of this new version is the use of a new language for designing bibliography styles. This language is close to XSLT and we give its manual as an annex.

Keywords bibliographies, multilingual features, BibTeX, bst, nbst, XML, XSLT, MlBibTeX.

1 Introduction

It is well known that a bibliography program should be associated with a text processor. If such a program is used for documents such as history articles, technical documentation, or research work, where many references may be cited, the role of a bibliography program is to search a database containing **bibliographical entries** for the citations throughout the document, sort them and arrange the information associated with each selected entry. In short, it has to build the 'References' section of the document, containing bibliographical **references**, which can be processed by the text processor at next run.

A bibliography program may look for *keys* surrounded by special markers within a source text, as does Tib [1]. Or it may use information included in auxiliary (.aux) files, as does BibTeX [16], most commonly used with (LA)TeX [14]. Here is an example of a bibliographical entry using BibTeX's syntax:

```
@BOOK{howard1967b,
        AUTHOR = {Robert~Ervin Howard},
        TITLE = {Conan the Conqueror},
        PUBLISHER = {Ace Books},
        ADDRESS = {New York, New York},
        NOTE = {Edited by L. Sprague de Camp},
        YEAR = 1967}
```

If the entry howard1967b is cited within a document, this information is put into an auxiliary file when LaTeX runs, so BibTeX can generate a .bbl file containing the corresponding reference. When LaTeX runs again, this reference will look like:

> [1] Robert Ervin Howard. *Conan the Conqueror*. Ace Books, New York, New York, 1967. Edited by L. Sprague de Camp.

according to the bibliography **style** chosen. Here and in the 'References' section of this article, we use a 'plain' style, that is, references are labelled with numbers, authors' last names are written using small capitals, and first names are not abbreviated. Other choices are possible: see [6, §13.2] for a survey of available bibliography styles.

Due to its conception, BibTeX has some limitations: its syntax is rough, bibliographic styles are written using an old-fashioned language [15], and multilingual bibliographies are supported only through workarounds. We personally missed this last point very much, thus we have put into action a new implementation of BibTeX, named MlBibTeX (for '**M**ulti**L**ingual BibTeX'), with many multilingual features. The first version (1.1) was described in [8]. But as we explained in [12], the new version described here (1.3) takes advantage of XML[1] and uses a new language, nbst, for '**n**ew **b**ibliography **st**yles', close to XSLT[2] [21].

This article aims to give a survey of all the new features introduced by MlBibTeX's present version. It is not a complete reference manual, but gives a good overview of the program. First, we describe the new syntactical features provided by MlBibTeX.[3] Then, we give some words about the implementation, showing the connection with XML and discussing two approaches for multiligual bibliographies. Then we explain how the information about languages is managed within our bibliography styles and show that nbst allows creators of bibliography styles to put them both into action. Last, a manual of elements and functions of the nbst language is given as an annex.

2 New syntactic features

Historically, we first added syntax for multilingual features [8]. Then we realised that some fields' values could be structured better with some new syntax. Here are the results of our choices.

2.1 Syntax for names

When BibTeX processes the value of an AUTHOR or EDITOR field, it divides a family name into four fields: *First* (for a first name), *von* (for a particle), *Last* (for a last name), and *Junior* and recognizes these components according to the following possible syntaxes [16, §4]:

(i) *First von Last*
(ii) *von Last*, *First*
(iii) *von Last*, *Junior*, *First*

As suggested by the cases used within this terminology, the words belonging to the *von* field are supposed to use only lowercase characters, whereas the

[1] E**X**tensible **M**arkup **L**anguage.
[2] E**X**tensible **S**tylesheet **L**anguage **T**ransformations.
[3] Let us note that 'old' .bib files are parsed successfully by MlBibTeX and give outputs comparable to BibTeX's, unless square brackets are used in field values.

```
@BOOK{howard1969,
        AUTHOR = {Robert Ervin Howard, abbr => R. with
                  first => Lyon Sprague, von => de, last => Camp, abbr => L. Sprague with
                  Lin Carter}
        TITLE = {Conan of {Cimmeria}},
        PUBLISHER = {Ace Books},
        ADDRESS = {New York, New York},
        NOTE = {[Titre de la traduction fran\c{c}aise : ‘‘Conan le Cimm\’{e}rien’’] ! french
                [Titel der deutschen \"{U}bersetzung: ‘‘Conan von Cimmerien’’] ! german}
        YEAR = 1969,
        LANGUAGE = english}
```

Figure 1: A multilingual entry using MlBibTeX's syntax.

words belonging to other fields are supposed to be capitalised. These rules are too restrictive: some particles may be capitalised, while some words belonging to a last name may be written using lowercase characters. Using additional braces solves some problems, but not all. In addition, BibTeX abbreviates a first name by retaining only the first letter of each word belonging to the *First* field, such letters being followed with a period character. That is sometimes incorrect: 'Jon L White' should be abbreviated to 'J. L White' or 'J. White', not to 'J. L. White'. First and middle American names are handled differently from one name to another. 'Robert Ervin Howard' is usually written down as 'Robert E. Howard', which becomes 'R. Howard' when the first name is abbreviated. In contrast, 'Henry Rider Haggard' is usually written down as 'H. Rider Haggard', and becomes 'H. R. Haggard' in styles where first names are abbreviated. In addition, several letters may be retained when abbreviating a non-English first name:

- in French, 'Charles Duits' is abbreviated to 'Ch. Duits', because the 'ch' group stands for one digraph ([ʃ]);
- likewise, 'Christian' is abbreviated to 'Chr.' in German.

Since Version 1.2 [10], MlBibTeX allows an explicit syntax for these fields and the abbreviation of a first name, if it is different from the 'standard way':

```
first => ..., von => ..., last => ...,
junior => ..., abbr => ...
```

The order of the keywords is irrelevant and some may be absent, provided that the last name is specified. For example:

```
first => Henry Rider, last => Haggard
```

where the *von* field is empty, and the abbreviation of the first name is standard, that is, 'H. R.' For a more complex example, see the specification of 'Lyon Sprague de Camp' in Figure 1. You can mix the 'old' and 'new' syntaxes, in which case a name is parsed like (i) if no comma occurs, like (ii) (resp. (iii)) if the number of commas not followed with a keyword is one (resp. two) and the keywords give additional information.[4] This is useful when we have to give a specific abbreviation for a first name: see the specification of 'Robert Ervin Howard' in Figure 1. In fact, this syntax is close to that for passing values inside a subprogram call in Ada [18, §6.4] and other languages.

When a name is not for a person but for an organisation, it is well known to BibTeX users that such an expression should be surrounded by additional braces:

```
EDITOR = {{\TUGboard 2003}}
```

so BibTeX considers it as a one-component name, this component being a *Last* part. However, this syntax poses a problem when a TeX command is used within such a name. In the given example, '`\TUGboard`' is viewed as an accent command: when the bibliography is sorted, the corresponding entry is alphabeticised as '2003'. MlBibTeX's new syntax allows the specification of both an organisation name and a key for sorting:

```
EDITOR = {org => \TUGboard 2003,
          sortingkey => TUG Board 2003}
```

As in BibTeX, co-authors are connected by the '`and`' keyword within .bib files. After one author or several successive co-authors, MlBibTeX allows the addition of *collaborators*, introduced by the '`with`' keyword. Figure 1 gives an example in MlBibTeX.[5]

[4] Nevertheless, defining any part of a name twice causes an error.

[5] Besides, the entry given in this figure allows us to emphasise the difference between co-authors and collaborators. In fact, L. Sprague de Camp and L. Carter sorted and arranged R. Howard's manuscripts after his death. So they are more 'collaborators' than co-authors. The entry `howard1967b`, given in the introduction, might be rewritten using this syntax, instead of using a `NOTE` field.

```
<book id="howard1969" language="english">
  <author>
    <name><personname><first abbr="R.">Robert Ervin</first><last>Howard</last></personname></name>
    <with/>
    <name>
      <personname>
        <first abbr="L. Sprague">Lyon Sprague</first><von>de</von><last>Camp</last>
      </personname>
    </name>
    <with/>
    <name><personname><first>Lin</first><last>Carter</last></personname></name>
  <title>
    Conan of <asitis>Cimmeria</asitis>
    <!--  asitis is for a group of words that should not be case-converted.  -->
  </title>
  <publisher>Ace Books</publisher>
  <year>1969</year>
  <address>New York, New York</address>
  <note>
    <group language="french">
      Titre de la traduction française : <emph emf="yes" quotedbf="yes">Conan le Cimmérien</emph>
    </group>
    <group language="german">
      Titel der deutschen Übersetzung: <emph emf="no" quotedbf="yes">Conan von Cimmerien</emph>
    </group>
  </note>
</inproceedings>
```

Figure 2: The entry given in Figure 1 viewed as an XML tree.

As in BibTeX, the '`others`' keyword can be used when additional names are left unspecified: '`and others`' and '`with others`' are allowed. In the bibliography of this article, reference [7] shows how such an entry using collaborators is formatted.

2.2 Syntax for multilingual features

In MlBibTeX's terminology, a **language identifier** is a non-ambiguous prefix of:

- an option of the babel package [2],
- or a multilingual package name such as french [5], german [17] or polski [4].[6]

The language of an entry is given by the `LANGUAGE` field, whose value is a language identifier (see Figure 1). This field defaults to '`english`'.

Here we only show the syntax we use for multilingual features included in .bib files; a more complete description can be found in [8], and more examples in [12]. In the following, 's', 's_1', ..., 's_n' are strings; n is a positive natural number; and 'l', 'l_1', ..., 'l_n' are language identifiers.

A **language change** is denoted by '`[s] : l`'. It is used for foreign words and in particular, it allows a text processor to hyphenate them correctly.

A **language switch without default language** is expressed by the following syntax:

$$\texttt{[}s_1\texttt{] ! }l_1\texttt{ ... [}s_n\texttt{] ! }l_n \qquad (1)$$

If there exists i ($0 \leq i \leq n$) such that the reference's language is equal to l_i, then Expression (1) yields s_i; otherwise, this expression is replaced by an empty string. In other words, this syntax is used for additional information that must be typeset in a particular language. For example, if we process the entry `howard1969` in French (resp. German), we can add the title of the French (resp. German) translation, as shown in the `NOTE` field in Figure 1.

A **language switch with default language** is expressed by the following syntax:

$$\texttt{[}s_1\texttt{] * }l_1\texttt{ ... [}s_n\texttt{] * }l_n \qquad (2)$$

This syntax is used for information that *must* be included, possibly in another language. If there exists i ($0 \leq i \leq n$) such that the reference's language is equal to l_i, then Expression (2) yields s_i; otherwise, this expression is replaced by the string associated

[6] This choice of a non-ambiguous prefix allows a language identifier to get access to several ways to process a language. For example, a language identifier set to `french` works with the frenchb option of the babel package as well as the french package.

with the language's entry if such a string exists, or by the string associated with the English language if not. For example, we could allow the publisher's address of the `howard1969` entry to use a Russian transliteration for a reference to this entry in Russian. Of course, this address is to be put in English otherwise. To do that, the `ADDRESS` field should be given such a value:

```
ADDRESS =
  {[New-York]
   [⟨Russian transliteration⟩] * russian}
```

Notice that '`[...]`', not followed with '`*`', '`!`' or '`:`' means '`[...] * 1`', where '`1`' is the language's entry.

2.3 Syntax for page numbers

In a `PAGES` field, MlBibTeX recognizes:

- a single page (one token): `{2003}`;
- the first and last pages (three tokens):

  `{2000--2003}` or `{2000-2003}`
- the first page and an unspecified number of following ones (two tokens): `{2003+}`;[7]
- some enumerated pages (five tokens in the example below): `{2000,2003,2005}`.

The tokens may or may not be separated by white-space[8] characters. In all the other cases, the value associated with this field is kept *verbatim* and appears as-is for any predefined bibliography style.

3 Implementation issues

MlBibTeX's first version [8] was written using C, for the sake of efficiency and portability. When we started implementation of the present version, we realised that we needed calls to *external functions* within our bibliography styles.[9] So we realised that it was preferable for our program to be written in a higher-level programming language. This way, the interface between bibliography styles and external functions would be designed better, so developers of new styles could write extensions in the source language more easily. We decided to develop a prototype in Scheme, with the features related to XML put into action by SXML[10] [13], an implementation of XML trees by means of Scheme expressions. Our nbst language, for bibliography styles, includes a `call` function (see Appendix B), that gives access to Scheme functions of MlBibTeX's library.

```
<nbst:bst version="1.3" id="plain"
   xmlns:nbst=
 "http://lifc.univ-fcomte.fr/~hufflen/mlbibtex">

 <!-- Reference-dependent approach: -->
 <nbst:param name="language" select="'*self*'"/>

 <!-- Root element grouping entries: -->
 <nbst:template match="mlbiblio">
   ...
 </nbst:template>

 ...
</nbst:bst>
```

Figure 3: Layout of a bibliography style file using nbst.

Parsing an MlBibTeX entry results in a representation of an XML tree in SXML; for example, the entry of Figure 1 is equivalent to the XML tree given in Figure 2, that is, if the SSAX[11] parser of SXML is applied to this XML tree, it yields the same result. Our XML trees modelling entries are conformant with a revised version of the DTD[12] sketched in [9]. They are rooted by the `mlbiblio` element, as suggested by the first template given in Figure 3.

In addition, SXML relies on functions extending the basic encoding of characters used in Scheme. These functions should allow Scheme programs to handle Unicode, but they are platform-dependent: some interpreters provide them, possibly partially, some do not. In practice, MlBibTeX can handle 8-bit latin1 encoding;[13] further development will be needed to adapt MlBibTeX to the whole of Unicode,[14] but the framework to do that is already present.

4 Multilingual approaches

As mentioned in [8], multilingual bibliographies can be organised with respect to two approaches, both of which can be put into action by MlBibTeX:

reference-dependent each reference of the document's bibliography is expressed using its own language: for example, the month name of a reference to a book written in English (resp. French, German, ...) is given in English (resp. French, German, ...);

[7] Such a specification is typeset as 'pp. 2003 ff.' in English-speaking bibliographies [3, §15.191].

[8] The whitespace characters are space, tab, newline, carriage return, and form feed.

[9] These external calls are used to manage information not included in .aux files. So it has to be directly extracted from .tex files.

[10] **S**cheme implementation of XML.

[11] Scheme implementation of SAX ('**S**imple API for XML').

[12] **D**ocument **T**ype **D**efinition (document markup model).

[13] [7, Table C.4] has more details about encodings.

[14] If you would like to use characters from non-Latin alphabets (e.g., Cyrillic characters), now put the LaTeX commands to produce them, rather than these characters themselves. A temporary situation, we hope.

```
<nbst:template match="author">
  <nbst:apply-templates/>
  <nbst:text>: </nbst:text>
</nbst:template>

<nbst:template match="name">
  <nbst:apply-templates/>
</nbst:template>
```

Figure 4: Formatting names in nbst.

document-dependent all references are expressed using the document's language, as far as possible.

5 The nbst Language

Most elements of nbst behave like their namesakes in XSLT. Figure 3 gives the general layout of a bibliography style and a representative example is given in Figures 4 & 5. The path expressions used in these figures are related to the tree given in Figure 2. Let us notice that some elements and attributes of are recognised by the nbst processor, but do not have any effect presently — they have been planned for future use of MlBibTeX, especially for generating XML documents[15] — this information is given in Appendix A. We assume that readers are quite familiar with XPath [20] and XSLT [21] — there exist some good introductory books about them, for example, [19] — so in this section we only explain how the language information is managed by the nbst processor.

Given a fragment of an entry viewed as a node (an XML subtree), its **current language** is the value of the `language` attribute if it exists, the value of the current language of its parent otherwise. The current language for an entry is the entry's language (see Section 2.2).

When templates are to be instantiated, the rule added to those inherited from XSLT is that a template with the `language` attribute has higher priority than the same template without it.[16] This rule overrides all the others. In particular, it applies if a template is invoked by name,[17] as well being applied if the current node matches the pattern of its `match` attribute.

When we begin to apply a bibliography style, the `language` attribute is associated with the document's language[18] (resp. the '`*self*`' value) according to the document-dependent (resp. reference-dependent) approach. When a template is to be invoked by name by means of such a statement:

```
<nbst:call-template name="..."/>
```

then we look for the current language. If this value is different from '`*self*`', we look for the named template with the `language` attribute set to this value if it exists. If not, the default named template, that is, without the `language` attribute, is invoked. The `use-language` attribute allows the redefinition of the current language; for example:

```
<nbst:call-template
    name="..." use-language="portuguese"/>
```

invokes a named template with the `language` attribute set to '`portuguese`' if such a template exists, its namesake without this attribute if not. The same rules applies for the `nbst:apply-templates` element:

```
<nbst:apply-templates
  select="S" use-language="finnish"/>
```

tries to find, for each node selected by the expression S, a template with the `language` attribute set to the right value (here, `finnish`) before instantiating the template without the `language` attribute. The same rule holds for templates with a `mode` attribute: given a set of templates with the same value associated with the `mode` attribute, we apply first the template with the right value for the `language` attribute, second the template without this attribute. As in XSLT [21, § 5.7], an `nbst:apply-templates` element with a `mode` attribute can only apply templates with the same value for this mode.

Using the '`*self*`' value is of little interest with an `nbst:call-template` element since the current node does not change when a template is invoked by its name. So the statement:

```
<nbst:call-template name="..."
                    use-language="*self*"/>
```

is equivalent to:

```
<nbst:call-template name="..."/>
```

unless the language of the template instantiated is not the current node's language. The statement:

```
<nbst:apply-templates
  select="S" use-language="*self*"/>
```

dispatches all the selected nodes w.r.t. their associated languages. It is equivalent to:

[15] In particular, we plan to investigate the generation of 'References' sections for DocBook documents [22].

[16] In fact, there are two levels of priority: the first is ruled by the `language` attribute, the second defined by XSLT, including the `priority` attribute.

[17] As a consequence, there can be several templates with the same name — which is an error in XSLT [21, §6] — provided that the values possibly associated with the different `language` attributes are pairwise-different.

[18] MlBibTeX tries to determine it as far as possible. Most often, it is the last option given to the babel package.

```
<nbst:template match="personname">
  <nbst:if test="first"><nbst:value-of select="first"/><nbst:text> </nbst:text></nbst:if>
  <nbst:if test="von"><nbst:value-of select="von"/><nbst:text> </nbst:text></nbst:if>
  <nbst:text>\textsc{</nbst:text><nbst:value-of select="last"/><nbst:text>}</nbst:text>
  <nbst:if test="junior">, Junior</nbst:if>
</nbst:template>

<nbst:template match="and">
  <nbst:choose>
    <nbst:when test="following-sibling::and or following-sibling::and-others">
      <nbst:text>, </nbst:text>
    </nbst:when>
    <nbst:otherwise>
      <nbst:text> </nbst:text><nbst:value-of select="$bbl.and"/><nbst:text> </nbst:text>
    </nbst:otherwise>
  </nbst:choose>
</nbst:template>

<nbst:template match="and-others">
  <nbst:text> </nbst:text><nbst:value-of select="$bbl.etal"/>
</nbst:template>
```

Figure 5: Formatting names with the nbst language (*continued*).

```
<nbst:for-each select="S">
  <nbst:apply-templates select="."
                        use-language="L"/>
</nbst:for-each>
```

where L is the current language of the current node. This expression is used for the `mlbiblio` element to build references in the reference-dependent approach.

As an example, the template given in Figure 6 is instantiated for this name:

```
AUTHOR = {[Zoltán Kodály] : hungarian}
```

6 Conclusion

Roughly speaking, we can consider that getting a bibliographical reference from an entry is a particular case of transformation — the same information, arranged differently. Thus, an XSLT-like language should be suitable for the task. In addition, our management of the information related to particular languages should ease the making of mutilingual bibliographies. At the time of writing, our program is in beta test and we have successfully rewritten a representative range of bibliography styles of BibTeX. So we think we are ready for public use and larger experiment.

7 Acknowledgements

Special thanks to Hans Hagen and Volker R. W. Schaa, who agreed to give the show associated with a preliminary version of this paper at the TUG 2003 conference. Thanks to Karl Berry and Barbara Beeton who proofread this revised and updated version.

References

[1] James C. Alexander: *Tib: A TeX Bibliographic Preprocessor*. Version 2.2, see `CTAN:biblios/tib/tibdoc.tex`. 1989.

[2] Johannes Braams: *Babel, a Multilingual Package for Use with LaTeX's Standard Document Classes*. Version 3.7. May 2002. `CTAN:macros/latex/required/babel/babel.dvi`.

[3] *The Chicago Manual of Style*. The University of Chicago Press. The 14th edition of a manual of style revised and expanded. 1993.

[4] Antoni Diller: *LaTeX wiersz po wierszu*. Wydawnictwo Helio, Gliwice. Polish translation of *LaTeX Line by Line* with an additional annex by Jan Jelowicki. 2001.

[5] Bernard Gaulle : *Notice d'utilisation du style french multilingue pour LaTeX*. Version pro V5.01. Janvier 2001. `CTAN:loria/language/french/pro/french/ALIRE.pdf`.

[6] Michel Goossens, Frank Mittelbach and Alexander Samarin: *The LaTeX Companion*. Addison-Wesley Publishing Company, Reading, Massachusetts. 1994.

[7] Michel Goossens and Sebastian Rahtz, with Eitan M. Gurari, Ross Moore and Robert S. Sutor: *The LaTeX Web Companion*. Addison-Wesley Longman, Inc., Reading, Massachusetts. May 1999.

```
<nbst:template match="personname" language="hungarian">   <!-- Here, the family name comes first. -->
  <nbst:text>\textsc{</nbst:text>
  <nbst:if test="von"><nbst:value-of select="von"/><nbst:text> </nbst:text></nbst:if>
  <nbst:value-of select="last"/><nbst:text>}</nbst:text>
  <nbst:if test="first"><nbst:text> </nbst:text><nbst:value-of select="first"></nbst:if>
  <nbst:if test="junior">, Junior</nbst:if>
</nbst:template>
```

Figure 6: Formatting Hungarian names with the nbst language.

[8] Jean-Michel Hufflen: "MlBibTeX: a New Implementation of BibTeX". In: *EuroTeX 2001* (pp. 74–94). Kerkrade, The Netherlands. September 2001.

[9] Jean-Michel Hufflen: "Multilingual Features for Bibliography Programs: from XML to MlBibTeX". In: *EuroTeX 2002* (pp. 46–59). Bachotek, Poland. April 2002.

[10] Jean-Michel Hufflen: "Towards MlBibTeX's Versions 1.2 & 1.3". MaTeX Conference. Budapest, Hungary. November 2002.

[11] Jean-Michel Hufflen: "Mixing Two Bibliography Style Languages". In: *LDTA 2003*, Vol. 82.3 of *ENTCS*. Elsevier, Warsaw, Poland. April 2003.

[12] Jean-Michel Hufflen: "European Bibliography Styles and MlBibTeX". *TUGboat*, Vol. 24, no. 3 (in process). EuroTeX 2003, Brest, France. June 2003.

[13] Oleg Kiselyov: "A Better XML Parser through Functional Programming". In: *4th International Symposium on Practical Aspects of Declarative Languages*, Vol. 2257 of *LNCS*. Springer-Verlag. 2002.

[14] Leslie Lamport: *LaTeX: A Document Preparation System. User's Guide and Reference Manual*. Addison-Wesley Publishing Company, Reading, Massachusetts. 1994.

[15] Oren Patashnik: "Designing BibTeX styles". February 1988. Part of BibTeX distributions.

[16] Oren Patashnik: "BibTeXing". February 1988. Part of BibTeX distributions.

[17] Bernd Raichle: *Die Makropakete „german" und „ngerman" für LaTeX 2ε, LaTeX 2.09, PlainTeX and andere darauf Basierende Formate*. Version 2.5. Juli 1998. Im Software LaTeX.

[18] S. Tucker Taft and Robert A. Duff, eds.: *Ada 95 Reference Manual. Language and Standard Libraries*. No. 1246 in LNCS. Springer-Verlag. International Standard ISO/IEC 8652:1995(E). 1995.

[19] Doug Tidwell: *XSLT*. O'Reilly & Associates, Inc. August 2001.

[20] W3C: *XML Path Language (XPath). Version 1.0*. W3C Recommendation. Edited by James Clark and Steve DeRose. November 1999. `http://www.w3.org/TR/1999/REC-xpath-19991116`.

[21] W3C: *XSL Transformations (XSLT). Version 1.0*. W3C Recommendation. Written by Sharon Adler, Anders Berglund, Jeff Caruso, Stephen Deach, Tony Graham, Paul Grosso, Eduardo Gutentag, Alex Milowski, Scott Parnell, Jeremy Richman and Steve Zilles. November 1999. `http://www.w3.org/TR/1999/REC-xslt-19991116`.

[22] Norman Walsh and Leonard Muellner: *DocBook: The Definitive Guide*. O'Reilly & Associates, Inc. October 1999.

Appendix A Elements of nbst

Hereafter, we describe each element of nbst. For each of them, we give its *syntax*: the attributes associated with it, and its content. For each attribute, we underline its name if it is required, and give the type of its possible values. When these values are enumerated, the default value is underlined.

The syntax is defined using regular expressions: the '|' sign means an alternative, '?' is used for an optional element, '*' (resp. '+') means zero (resp. one) or more occurrences of an element.

Here are the type identifiers used throughout this section:

CDATA for '**C**haracter **DATA**', that is, literal data characters without '<', '>', '&';[19]

char literal character;

expr analogous to an XPath expression;

id *unique* identifier for a resource;

lg-expr expression that results in either a non-ambiguous prefix of available languages or the '*self*' keyword;

name simple identifier;[20]

[19] As in XML, use the entities '<', '>', '&' for these characters.

[20] '*name*' is used instead of 'qualified name' within XSLT since Version 1.3 does not allow namespaces, except for nbst.

nmtoken whitespace-free sequence of characters;

number *constant* number;

pattern expression allowed within the `match` attribute of the `nbst:template` element;

template any (possibly empty) sequence of nbst elements, except for top-level ones;

top-level-elt element allowed at the top level;

uri-ref now a simple identifier.[21]

Plurals denote non-empty sequences whose elements are separated by whitespace characters: for example, '*names*' is for a non-empty sequence of objects each of type '*name*'.

`<nbst:accumulate>`

Synt.:
```
<nbst:accumulate>
  template
</nbst:accumulate>
```

Pushes the result of *template* onto the stack used when we process a bst function (see [11] for more details). Several `nbst:accumulate` elements can be given sequentially, but they cannot be nested.

`<nbst:apply-templates>`

Synt.:
```
<nbst:apply-templates
    select=expr mode=name
    use-language=lg-expr>
  (nbst:with-param |
   nbst:sort)*
</nbst:apply-templates>
```

Processes the node set selected by the value of the `select` attribute, or all the children of the current node by default. The selected node set is processed in document order, unless a sorting specification is present. About the attributes `mode` and `use-language`, see Section 5.

`<nbst:attribute>`

Synt.:
```
<nbst:attribute name=name>
  template
</nbst:attribute>
```

Recognised, but does not have any effect, like `nbst:attribute-set` and `nbst:element`. See Section 5.

`<nbst:attribute-set>`

Synt.:
```
<nbst:attribute-set
    name=name
    use-attribute-sets=names>
  nbst:attribute*
</nbst:attribute-set>
```

See `nbst:attribute`.

[21] True ***U****niform* ***R****esource* ***I****dentifiers*, in the sense of XML, will be allowed in a future version.

`<nbst:bst>`

Synt.:
```
<nbst:bst id=id version=number>
  top-level-elt*
</nbst:bst>
```

Root element of a bibliography style. The only version number presently recognised is 1.3.

`<nbst:call-template>`

Synt.:
```
<nbst:call-template
    name=name
    use-language=lg-expr>
  nbst:with-param*
</nbst:call-template>
```

Invokes a template by name by means of the required `name` attribute. See Section 5 about the `use-language` attribute.

`<nbst:choose>`

Synt.:
```
<nbst:choose>
  nbst:when+ nbst:otherwise?
</nbst:choose>
```

Each of the `nbst:when` elements is tested in turn, until reaching an element whose test is *true*, in which case the content is instantiated. If no such element exists, then the content of the `nbst:otherwise` element is instantiated if it exists, otherwise nothing is created.

`<nbst:comment>`

Synt.:
```
<nbst:comment>
  template
</nbst:comment>
```

Puts the result of *template* as a comment. In practice, now used to write lines beginning with '%' in LaTeX mode.

`<nbst:copy>`

Synt.:
```
<nbst:copy
    use-attribute-sets=names>
  template
</nbst:copy>
```

Copies the current node at the first level onto the result. The `use-attribut-sets` attribute does not have any effect presently.

`<nbst:copy-of>`

Synt.: `<nbst:copy-of select=expr/>`

Copies the whole of the node set selected by the required `select` attribute.

`<nbst:decimal-format>`

Synt.:
```
<nbst:decimal-format
    name=name
    decimal-separator=char
    grouping-separator=char
    infinity=cdata
    minus-sign=char NaN=cdata
```

```
percent=char per-mille=char
zero-digit=char digit=char
pattern-separator=char/>
```

Declares a decimal format, which rules the interpretation of a format pattern used by the `format-number` function. If there is a `name` attribute, then this element declares a named decimal format; otherwise, it declares the default decimal format. Here are the other attributes:

- `decimal-separator` specifies the character used for the decimal sign, defaults to the period character ('.');
- `grouping-separator`: the character used as a grouping (e.g., thousands) separator, defaults to ',';
- `infinity`: the identifier used to represent infinity, defaults to 'Infinity';
- `minus-sign`: the character used as the default minus sign, defaults to '-';
- `NaN`: the identifier used to represent a value that should be a number but is not, defaults to 'NaN' ('**N**ot **a** **N**umber');
- `percent` and `per-mille`: the two characters used as percent and per-mille signs ('%' and '‰'); in LaTeX mode, default to the command producing them ('\%' and '\textperthousand'[22]);
- `zero-digit`: a character always replaced by a digit, defaults to '0';
- `digit`: a character used for a digit, left blank for a missing digit, defaults to '#';
- `pattern-separator`: the character used to separate sub-patterns for positive and negative patterns, defaults to ';'.

`<nbst:element>`

Synt.:
```
<nbst:element
    name=name
    use-attribute-sets=names>
  template
</nbst:element>
```

See `nbst:attribute`.

`<nbst:for-each>`

Synt.:
```
<nbst:for-each select=expr>
  nbst:sort* template
</nbst:for-each>
```

template is instantiated for each node selected by the required `select` expression, which must evaluate to a node set. The selected nodes are processed in document order, unless a sorting specification is present.

[22] Notice that this command can be used with the Cork encoding, that is, the T1 option of the fontenc package.

`<nbst:if>`

Synt.:
```
<nbst:if test=expr>
  template
</nbst:if>
```

If the evaluation of the `test` attribute results in *true*, then *template* is instantiated; otherwise, nothing is created.

`<nbst:include>`

Synt.: `<nbst:include href=uri-ref/>`

Includes elements belonging to another nbst or bst file, identified by the `href` attribute. Allowed as a top-level element only.

`<nbst:key>`

Synt.:
```
<nbst:key name=name
          match=pattern
          use=expr/>
```

Recognised but does not have any effect.

`<nbst:message>`

Synt.:
```
<nbst:message
    terminate=("yes" | "no")>
  template
</nbst:template>
```

Displays the result of *template* as a message. If the `terminate` attribute has the value 'yes', then the program terminates after displaying the message.

`<nbst:number>`

Synt.:
```
<nbst:number
    level=("single" |
           "multiple" | "any")
    count=pattern from=pattern
    value=expr format=cdata
    language=lg-expr
    letter-value=
    ("alphabetic" |
     "traditional")
    grouping-separator=char
    grouping-size=number/>
```

Puts a formatted number. The number may be specified by means of the `value` attribute, in which case the expression is evaluated and the `number` and `round` functions are applied to the resulting object. If no `value` attribute is specified, then the inserted number is based on the position of the current node, controlled by the following attributes:

- `level` specifies which levels of the source tree should be considered;
- `count` attribute is a pattern that specifies what nodes should be counted at those levels: if it is unspecified, it defaults to the

pattern matching any node with the same node type as the current node;

- `from`: a pattern that specifies where counting starts.

The `format` attribute is split into alphanumeric and non-alphanumeric characters. The former are formats for numbers:

- '`1`' for `1`, `2`, ...
- '`i`' (resp. '`I`') for `i`, `ii`, ... (resp. `I`, `II`, ...)
- '`a`' (resp. '`A`') for `a`, `b`, ... (resp. `A`, `B`, ...), the `language` attribute being used to determine the alphabetical order.

The latter are copied *verbatim* onto the formatted string. Consult `nbst:decimal-format` about the `grouping-separator` attribute. The `grouping-size` attribute specifies the size of the grouping, defaulting to 3. If only one of these two attributes is specified, then it is ignored. The `letter-value` attribute does not have any effect.

`<nbst:otherwise>`

Synt.:
```
<nbst:otherwise>
  template
</nbst:otherwise>
```

See `nbst:choose`.

`<nbst:output>`

Synt.:
```
<nbst:output
    method=("LaTeX" | "xml" |
            "html" | "text")
    version=nmtoken
    encoding=cdata
    omit-xml-declaration=
    ("yes" | "no")
    standalone=("yes" | "no")
    doctype-public=cdata
    doctype-system=uri-ref
    cdata-section-elements=
    names
    indent=("yes" | "no")
    media-type=cdata/>
```

Only allowed as a top-level element. Allows bibliography style writers to specify how they wish the result to be output. Presently, the values allowed for the `method` attribute are:

- '`LaTeX`', for LaTeX output;
- '`xml`' (resp. '`html`'), for XML (resp. HTML) output; however, do not forget that, as with XSLT, the output for an HTML file must be written according to XHTML[23] conventions;
- '`text`', for verbatim text output.

Other attributes:

- `version` specifies the version of the output method,
- `encoding`: the character encoding to be used;
- `omit-xml-declaration`: whether or not the XML declaration should be output;
- the other attributes do not have any effect.

`<nbst:param>`

Synt.:
```
<nbst:param name=name
            select=expr>
  template
</nbst:param>
```

Used at the top level to define an external parameter or within a template rule to specify a local parameter. The `select` attribute gives a default value. When this attribute is absent, the default value is given by instantiating *`template`* if it is not empty. If this parameter is not given a default value, nbst pops the stack used when we process a bst function; if this stack is empty, the value given to the parameter is the empty string.

`<nbst:sort>`

Synt.:
```
<nbst:sort
    select=expr
    language=lg-expr
    data-type=
    ("text" | "number")
    order=("ascending" |
           "descending")
    case-order=("upper-first" |
                "lower-first")/>
```

Used as a child of an `nbst:apply-templates` or `nbst:for-each` element. The first occurrence specifies the primary sort key, the second occurrence the secondary sort key used for elements left unsorted, and so on. The key is given by the `select` attribute, which defaults to '`.`'. This expression is applied to each node of the current set, and the result is converted into a string or a number, w.r.t. the value of the `data-type` attribute. In addition:

- `order` can be ascending or descending;
- `language`: the sort keys' language;
- `data-type`: the sort keys' data type:
  - '`text`' means that they should be lexicographically sorted in the culturally correct way for the current language,
  - '`number`' specifies a numerical sort, in which case `language` is ignored;

[23] E**X**tensible **H**yper**T**ext **M**arkup **L**anguage.

- the possible values for `case-order` apply when `data-type` is '`text`', and specifies that upper-case letters should sort before lower-case letters or *vice-versa*. The default value is language-dependent.

`<nbst:template>`

Synt.:
```
<nbst:template
    match=pattern name=name
    language=lg-expr
    priority=number mode=name>
  nbst:param* template
</nbst:template>
```

Defines a template rule. The `match` attribute is a pattern that identifies the source node to which the rules apply. The `match` attribute is required unless a `name` attribute is given, but both attributes can be specified. It is an error for the value of the `match` attribute to contain a reference to a variable. When such a rule is applied, *template* is instantiated.

Templates can be invoked by name, in which case the `match` attribute has no effect; likewise with the `name` attribute if the template is invoked by an `nbst:apply-templates` element. The role of the attributes `language`, `mode` and `priority` is explained in Section 5.

`<nbst:text>`

Synt.:
```
<nbst:text
    disable-output-escaping=
    ("yes" | "no")>
  cdata
</nbst:text>
```

Copies its content *verbatim* onto the output. The `disable-output-escaping` attribute does not have any effect.

`<nbst:variable>`

Synt.:
```
<nbst:variable name=name
               select=expr>
  template
</nbst:variable>
```

Analogous to `nbst:param`, but the value associated with a variable cannot be redefined by an element such as `nbst:with-param`.

`<nbst:value-of>`

Synt.:
```
<nbst:value-of
    select=expr
    disable-output-escaping=
    ("yes" | "no")/>
```

The value of the required `select` attribute is evaluated and the resulting object is converted to a string. The `disable-output-escaping` attribute does not have any effect.

`<nbst:warning>`

Synt.:
```
<nbst:warning>
  template
</nbst:warning>
```

Equivalent to `nbst:message` with `terminate` set to '`no`'.

`<nbst:when>`

Synt.:
```
<nbst:when test=expr>
  template
</nbst:when>
```

See `nbst:choose`.

`<nbst:with-param>`

Synt.:
```
<nbst:with-param
    name=name select=expr>
  template
</nbst:with-param>
```

Passes values to parameters before instantiating templates. The required `name` attribute specifies the name of the parameter, its value is specified in the same way as for `nbst:param`. The current node and node list used for computing the value are the same as for the element within which it can occur (`nbst:apply-templates` or `nbst:call-template`).

Appendix B Functions associated with our paths

We begin this section by describing the types used within the functions associated with our paths. As in XPath, we allow some type conversions. So, for each type, we mention which other types can be converted into it.

`boolean` is for the truth values: *true* and *false*. A node set is viewed as *false* if it is empty, as *true* otherwise. Likewise a string. A number is viewed as *false* if it is equal to zero, *true* otherwise.

`node-set` A node set belonging to the tree of bibliographical entries. A string can be converted into a one-element node set if it is a well-formed XML text, otherwise the result is an empty node set. A boolean or numerical value can be converted into a text node.

`number` When applied to integers, functions using numbers return integer results as far as possible, real numbers otherwise. A string can be converted into a number, provided the characters it contains form a number, possibly surrounded by whitespace characters:

`"␣-273.15"` is a number,
`"-␣273.15"` is not.

If such a conversion fails, the result is `NaN`. If `NaN` is used instead of a number as an argument of a numeric function, the result is `NaN`.

`string` Boolean and numbers can be converted into strings. So can the values for numeric errors, `Infinity` and `NaN`. Node sets too, in which case an attribute node is converted into its associated value, whereas an element node is converted into the concatenated values of all the text nodes inside it.

Throughout this section, '*n*', '*ns*', '*s*' denote variables of type `number`, `node-set`, `string` respectively, whereas '*x*' is for an expression of any type. If several variables of the same type are needed, we use indices. Some functions can be applied to any number of arguments, in which case the additional optional arguments are denoted by '...'. As in XPath, some arguments can be omitted, in which case the current node set is passed: we denote this behaviour by a question mark ('`?`'). For each function, we give the type of its result, a template of its use and a short description of its behaviour.

`!=`

Use: `boolean` x_1 `!=` x_2

Returns *true* if x_1 and x_2 are distinct objects, *false* otherwise.[24]

`*`, `+`, `-`

Use: `number` n_1 `*` n_2 (resp. n_1 `+` n_2, n_1 `-` n_2)

Returns $n_1 * n_2$ (resp. $n_1 + n_2$, $n_1 - n_2$).

`<`, `<=`

Use: `boolean` n_1 `<` n_2 (resp. n_1 `<=` n_2)

Returns *true* if $n_1 < n_2$ (resp. $n_1 \leq n_2$), *false* otherwise.[24]

`=`

Use: `boolean` x_1 `=` x_2

Returns *true* if:

- x_1 and x_2 are the same object,
- or have a common element if x_1 or x_2 is a node set;

returns *false* otherwise.[24]

`>`, `>=`

Use: `boolean` n_1 `>` n_2 (resp. n_1 `>=` n_2)

Returns *true* if $n_1 > n_2$ (resp. $n_1 \geq n_2$), *false* otherwise.[24]

`abbreviate`

Use: `string abbreviate(`*s*`)`

Assuming that *s* is a first name, returns its abbreviation. If an *ad hoc* abbreviation has been specified by means of the `abbr` keyword, returns it. Otherwise, *s* is abbreviated in a standard way, that is, the initials and the hyphen character are retained:

```
abbreviate("John Fitzgerald)"
                              yields "J. F."
abbreviate("Paul-Loup") ..... "P.-L."
```

`abs`

Use: `number abs(`*n*`)`

Returns the absolute value of *n*.

`and`

Use: `boolean` b_1 `and` b_2

Returns *true* if b_1 and b_2 are both *true*, *false* otherwise.

`boolean`

Use: `boolean boolean(`*x*`)`

Converts *x* to a boolean *true* or *false* value.

`call`

Use: `string call(`s_1`,`s_2`,...)`

Calls s_1, a function included in MlBibTeX's library, with the arguments s_2, ... The s_1 function must return a string which is the result of the `call` function. In practice, this function is used by the multilingual interface.

`ceiling`

Use: `number ceiling(`*n*`)`

Returns the smallest integer that is greater than or equal to *n*.

`concat`

Use: `string concat(`s_1`,`s_2`,...)`

Returns the concatenation of the values of the passed arguments.

`contains`

Use: `string contains(`s_1`,`s_2`)`

Returns *true* if s_1 contains s_2, *false* otherwise.

`count`

Use: `number count(`*ns*`)`

Returns the number of nodes in *ns*.

`current`

Use: `node-set current()`

Returns the current node as a node set.

`div`

Use: `number` n_1 `div` n_2

Divides n_1 by n_2. If n_2 is equal to zero, this operation results in `Infinity` — this value is not a string.

`false`

Use: `boolean false()`

Returns the *false* value.

`firstcapitalize`

Use: `string firstcapitalize(`*s*`)`

Converts *s* to all lowercase except for the first word, which is capitalised.

[24] Notice that `NaN != NaN` yields *true*, whereas `NaN` *op* `NaN` yields *false* if *op* $\in \{<, <=, =, >, >=\}$.

`floor`

Use: `number floor(n)`

Returns the largest integer that is less than or equal to *n*.

`format-number`

Use: `number format-number(n,s1,s2?)`

Formats *n* according to the specifications of s_1 (see `nbst:decimal-format`) and the name s_2.

`generate-newly`

Use: `string generate-newly(s1,s2,ns?)`

Returns a unique string associated with the first node of *ns*. If s_1 is not empty, it is used as result's prefix. If s_2 is not empty, it must be a format used for numbers (see the description of the `format` attribute of `nbst:number`) and is used to generate result's suffixes.

`id`

Use: `node-set id(x)`

Returns the element node with an ID-type equal to the value of *x*. This function is useful when we are looking for an entry.

`is-boolean`

Use: `boolean is-boolean(x)`

Returns *true* if *x* is a boolean value, *false* otherwise.

`is-defined`

Use: `boolean is-defined(s)`

Returns *true* if *s* is the name of a parameter or variable bound to a value, *false* otherwise.

`is-node-set`

Use: `boolean is-node-set(x)`

Returns *true* if *x* is a (possibly empty) node set, *false* otherwise.

`is-number`

Use: `boolean is-number(x)`

Returns *true* if *x* is a number, *false* otherwise.

`is-string`

Use: `boolean is-string(x)`

Returns *true* if *x* is a string, *false* otherwise.

`key`

Use: `node-set key(s,x)`

Not implemented presently, so always returns an empty node set.

`last`

Use: `integer last()`

Returns the number of nodes in the current node set.

`local-name`

Use: `string local-name(ns?)`

Returns the name of the first node of *ns?*.

`lowercase`

Use: `string lowercase(s)`

Converts *s* completely to lowercase.

`mod`

Use: `number n1 mod n2`

Returns the remainder after dividing n_1 by n_2. The result always has the sign of n_1. If n_2 is equal to zero, the result is `NaN`.

`name`

Use: `string name(ns?)`

Returns the name of the first node of *ns*.[25]

`node-set`

Use: `node-set node-set(x)`

Converts *x* to a node set.

`normalize-space`

Use: `string normalize-space(s)`

Returns the whitespace-normalised value of *s*, that is, *s* is stripped of leading and trailing whitespace characters, and multiple consecutive occurrences of whitespace characters are replaced by a single space.

`not`

Use: `boolean not(b)`

Returns *true* (resp. *false*) if *b* is *false* (resp. *true*).

`number`

Use: `number number(x)`

Converts *x* to a numerical value.

`or`

Use: `boolean b1 or b2`

Returns *true* if b_1 or b_2 is *true*, *false* otherwise.

`position`

Use: `integer position()`

Returns the ordinal position of the context node within the context node set. These positions are counted starting from one, as in XPath.

`round`

Use: `number round(n)`

Returns the integer nearest in value to *n*. If *n* has a decimal portion of exactly .5, rounds up.

`starts-with`

Use: `boolean starts-with(s1,s2)`

Returns *true* if s_1 begins with s_2, *false* otherwise.

`string`

Use: `string string(x)`

Converts *x* to a string.

[25] Presently, the `name` and `local-name` functions return the same result since Version 1.3 does not allow namespaces.

`string-length`
Use: `number string-length(`*s*`)`
Returns the number of characters in *s*.

`substring`
Use: `string substring(`*s*`,`n_1`,`n_2`)`
Returns the portion of *s* starting at character n_1, for a length of n_2 characters.

`substring-after`
Use: `string substring-after(`s_1`,`s_2`)`
Returns the portion of s_1 following s_2.

`substring-before`
Use: `string substring-before(`s_1`,`s_2`)`
Returns the portion of s_1 preceding s_2.

`sum`
Use: `number sum(`*ns*`)`
Returns the sum of all nodes in *ns* after converting each to a number.

`translate`
Use: `string translate(`s_1`,`s_2`,`s_3`)`
Replaces any individual characters appearing in both s_1 and s_2 with corresponding characters in s_3.

`true`
Use: `boolean true()`
Returns the *true* value.

`uppercase`
Use: `string uppercase(`*s*`)`
Converts *s* completely to uppercase.

Appendix C Comparison with XPath and XSLT

Here we sum up the differences between XPath and XSLT on the one hand, and nbst on the other. These languages are close to each other, so learning nbst is easy if you know XPath and XSLT.

C.1 nbst vs XSLT

The corresponding element of the `xsl:stylesheet` element in XSLT is `nbst:bst` in nbst. For the sake of compatibility with the bst language of BibTeX, we added the `nbst:warning` element, but it can be viewed as a particular case of `nbst:message`, close to `xsl:message`.

- XSLT elements without equivalent in nbst:

`xsl:apply-imports`	`xsl:namespace-alias`
`xsl:fallback`	`xsl:preserve-space`
`xsl:import`	
`xsl:processing-instruction`	
	`xsl:strip-space`

- nbst element without equivalent in XSLT:

`nbst:accumulate`

C.2 XPath vs nbst paths

- XPath functions not included in nbst:

`document`	`namespace-uri`
`element-available`	`system-property`
`function-available`	`unparsed-entity-uri`
`lang`	

- Additional functions in nbst:

`abbreviate`	`is-defined`	`lowercase`
`call`	`is-node-set`	`node-set`[26]
`firstcapitalize`	`is-number`	`uppercase`
`is-boolean`	`is-string`	

- Close, but not identical functions:

(XSLT) `generate-id` ∼ `generate-newly` (nbst)

◇ Jean-Michel Hufflen
LIFC (FRE CNRS 2661)
University of Franche-Comté
16, route de Gray
25030 Besançon Cedex
France
`hufflen@lifc.univ-fcomte.fr`
`http://lifc.univ-fcomte.fr/~hufflen`

[26] This function is provided by some XSLT processors, but has not been included in the 'official' specification of XSLT [21]. It belongs to the additional functions of the EXSLT ('**E**xtensions to XSLT') project (for more details, see the Web page `http://www.exslt.org`).

Hints & Tricks

Glisterings

Peter Wilson

> Not all that tempts your wand'ring eyes
> And heedless hearts, is lawful prize;
> Nor all, that glisters, gold.
>
> *Ode to a Favourite Cat*
> Thomas Gray

The aim of this column is to provide odd hints or small pieces of code that might help in solving a problem or two. I have learnt to my cost that the quickest way to get a correct answer to a question on the `comp.text.tex` (`ctt`) newsgroup is to give an incorrect answer. I hope that the ideas below

always work but as the column title implies, there might be some dross among the nuggets.

Corrections, suggestions, and contributions will always be welcome.

One of the less frequently asked questions on `ctt` is whether there is a package for typesetting forms, and the answer is 'no'. Nevertheless forms are still typeset via LaTeX.

> Little boxes on the hillside, little boxes made of ticky tacky.
> Little boxes, little boxes, little boxes all the same.
>
> *Little boxes*
> Malvina Reynolds, 1961
> (Popularised by Pete Seeger)

1 Tick boxes

One common component of a form is the tick box.
Can you produce a tick box? Yes ☐ No ☐
Can I produce a tick box? Yes ☒ No ☐

The empty boxes above were produced by the `\tickbox` macro defined below, and the `\xbox` command produced the checked box.

```
\newcommand*{\tickbox}{{\fboxsep 0pt%
  \framebox[\height]{\vphantom{M}}}}
\newcommand*{\xbox}{{\fboxsep 0pt%
  \framebox[\height]{\vphantom{M}$\times$}}}
```

If you want a bit bigger box, like ☐, you could use this larger `\Tickbox` instead of `\tickbox`.

```
\newcommand*{\Tickbox}{\framebox{\phantom{M}}}
```

The length `\fboxsep` is the space between the frame of a `\framebox` and its contents, so you can adjust the size of this kind of box by changing `\fboxsep` or by using a different phantom character.

Or you might prefer ☐, from

```
\newcommand*{\TickBox}{{\fboxsep 0pt%
  \fbox{\rule{0em}{1em}\rule{1em}{0em}}}}
```

This definition uses invisible rules (a `\rule` with zero height or width cannot be seen) to control the size of the tightly fitting `\fbox`. In this case, as the rules are the same length as the box is square.

2 Blanks, dashes and rules

Along the lines of checkboxes, another regular component of a form is _____.

The last sentence ended with `\hrulefill{}.` to give a rule that stretched to margin. You can use `\hrulefill` more than once in a line, such as in the next line where it is used twice.
Last name: _____ First: _____

Or, you may want to have a rule of a particular length. The rule in the following line is just long enough for the word 'something' to be typeset:
Put _____ here.
Put something here.

The previous two lines were input as:

```
Put \underline{\phantom{something}} here. \\
Put something here.
```

Here is a variety of rules and blanks. In each case I've shown the code of the interesting part of the sentence above the typeset result.

1. `... in \rule{10mm}{0.4pt} the ...`
   Fill in _____ the blank.
2. `... in \hrulefill{} the ...`
   Fill in _____ the blank.
3. `... in \xfill[0.5ex] the ...`
   Fill in _____ the blank.
4. `... in \srule{something} the ...`
   Fill in _____ the blank.
5. `... in \srule[0.5ex]{something} the ...`
   Fill in _____ the blank.
6. `... in \phantom{something} the ...`
   Fill in the blank.

The macros used above are part of LaTeX with the exceptions of `\xfill` and `\srule` which are defined below.

The `\xfill[`⟨*len*⟩`]` macro is like `\hrulefill` in that it draws a rule in the available space but you can use the optional ⟨*len*⟩ length argument to raise or lower the rule with respect to the baseline.

```
\newcommand*{\xfill}[1][0pt]{%
  \cleaders
    \hbox to 1pt{\hss
      \raisebox{#1}{\rule{1.2pt}{0.4pt}}%
      \hss}\hfill}
```

`\srule[`⟨*len*⟩`]{`⟨*text*⟩`}` draws a rule the same length as the ⟨*text*⟩, but does not typeset the ⟨*text*⟩. You can use the optional ⟨*len*⟩ argument to alter the height of the rule with respect to the baseline.

```
\newcommand*{\srule}[2][0pt]{%
  \setbox0\hbox{#2}%
  \rule[#1]{\wd0}{0.4pt}}
```

The `\rule` command has an optional argument, which is the amount to raise (lower) the rule from its normal position at the baseline. LaTeX provides two dashes, the en-dash (–), input as `--` and the em-dash (—), input as `---`. En-dashes are used as the separator in a number range, like 2–4. Depending on your country's typesetting tradition an en-dash or an em-dash may be used instead of a comma as a phrase separator in normal text. Longer dashes may be used to indicate that something is missing; a 2em dash (——) for missing letters in a word and a 3em dash (———) to indicate a missing word. You can use `\rule` as a basis for longer dashes; for instance

Customs Declaration **CD 44**
May be opened officially

☒ Gift ☐ Commercial sample
☐ Documents ☐ Other

Quantity and detailed description of contents	Weight lb.	oz.	Value
Toy			15
Scarf			12
For commercial items only *If known,* HS tariff number and country of origin of goods	Total Weight		Total Value

I, the undersigned, whose name and address are given on the item, certify that the particulars given in this declaration are correct and that this item does not contain any dangerous article or articles prohibited by legislation or by postal or customs regulations.

Date and sender's signature

PS Form **1234**, March 2004

```
\newcommand*{\iiemdash}{%   2em dash
  \rule[0.5ex]{2em}{0.4pt}}
\newcommand*{\iiiemdash}{%  3em dash
  \rule[0.5ex]{3em}{0.4pt}}
```

3 Forms

I have found that often the easiest way for me to define a form is to use the `picture` environment as this lets me place things just where I want them. Here is a possibly boring example for a customs declaration form; the real form is about 10% smaller than the example.

```
\newcommand{\form}{%
\setlength{\unitlength}{1mm}
\begin{picture}(79,80)
\sffamily \scriptsize \thicklines
\put(0,0){\line(1,0){80}}
\put(0,5){\line(1,0){80}}
\put(2,4){\makebox(0,0)[tl]{\normalsize PS Form
          \textbf{1234}, March 2004}}
\put(0,14){\line(1,0){80}}
\put(2,13){\makebox(0,0)[tl]{Date and sender's
           signature}}
\put(0,26){\line(1,0){80}}
\put(2,25){\makebox(0,0)[tl]{%
  \begin{minipage}{76mm}
  I, the undersigned, ... regulations
  \end{minipage}}}
\put(0,30){\line(1,0){48}}
\put(0,39){\line(1,0){80}}
\put(2,38){\makebox(0,0)[tl]{%
  \begin{minipage}{44mm}
  \textbf{For commercial items only} \\
  \textsl{If known,} ... \end{minipage}}}
\put(56,38){\makebox(0,0)[t]{Total Weight}}
\put(72,38){\makebox(0,0)[t]{Total Value}}
\put(0,53){\line(1,0){80}}
\put(2,52){\makebox(0,0)[tl]{%
  \begin{minipage}{40mm}
  \CONT \end{minipage}}}
\put(0,60){\line(1,0){80}}
\put(2,59){\makebox(0,0)[tl]{%
  \begin{minipage}{40mm}
  Quantity ... \end{minipage}}}
\put(49,59){\makebox(0,0)[tl]{%
  \begin{minipage}{14mm}
  \hfill Weight \hfill \mbox{}\\
  lb. \hfill oz. \end{minipage}}}
\put(72,58){\makebox(0,0)[t]{Value}}
\put(65,52){\makebox(0,0)[tl]{%
  \begin{minipage}{14mm}
  \CVAL \end{minipage}}}
\put(0,68){\line(1,0){80}}
\put(14,61){\makebox(0,0)[bl]{\DBX\ Documents}}
\put(14,66){\makebox(0,0)[tl]{\GBX\ Gift}}
\put(34,61){\makebox(0,0)[bl]{\OBX\ Other}}
\put(34,66){\makebox(0,0)[tl]{\CBX\ Commercial
                              sample}}
\put(0,80){\line(1,0){80}}
\put(2,79){\makebox(0,0)[tl]{%
  \begin{minipage}{76mm}\normalsize
  \textbf{Customs Declaration} ...
  officially  \end{minipage}}}
%% vertical lines
\put(48,26){\line(0,1){34}}
\put(64,26){\line(0,1){34}}
\thinlines
\put(56,26){\line(0,1){9}}
\put(56,39){\line(0,1){14}}
\end{picture}
\setlength{\unitlength}{1pt}
}% end of \form
```

The variable parts of the form (i.e., the non-commercial answers) are represented by the uppercase commands, which have to be defined for any specific instance of the `\form`. For the example shown the code to complete and display it is:

```
\let\GBX\xbox    \let\DBX\tickbox
\let\OBX\tickbox \let\CBX\tickbox
\newcommand{\CONT}{\normalsize\rmfamily
Toy \\ Scarf}
\newcommand{\CVAL}{\normalsize\rmfamily
\centering 15 \\ 12}

\begin{figure}
\centering
\form
\end{figure}
```

4 Letters

I received the following[1] from Michael Barr regarding string comparisons. Perhaps someone can help?

Following your column in the recent issue of *TUGboat* [1] I have a problem in string comparisons that I don't think is solvable. At least I couldn't. Suppose you want to decide if an argument ultimately expands to nothing. Or if two arguments have the same ultimate expansion. What I was finally led to was similar to your `\strcfstr` on page 340, [Ed. reproduced here]

```
\newif\ifsame
\newcommand{\strcfstr}[2]{%
  \samefalse
  \begingroup
    \def\1{#1}\def\2{#2}%
    \ifx\1\2\endgroup \sametrue
    \else   \endgroup
    \fi}
```

except with `\def` replaced by `\edef`. This worked until the day that the argument (I was actually testing for being empty, but the difficulty is the same) happened to be a matrix. At this point, I got a curious error message about misplaced &. It turns out that while you can put a matrix (or any alignment) into a `\def`, you cannot put one into an `\edef`. This is a built-in 'feature' (pronounced 'bug') of TEX, one that will not be repaired and one that it is apparently impossible to overcome.

Michael Barr

References

[1] Peter Wilson. Glisterings. *TUGboat*, 22(4):339–341, December 2001.

◇ Peter Wilson
18912 8th Ave. SW
Normandy Park, WA 98166
USA
herries.press@earthlink.net

[1] I have exercised editorial privilege on the original.

The Treasure Chest

This is a list of packages posted to CTAN from July 2002 through December 2003, with descriptive text pulled from the announcement or researched and edited for brevity. Please inform us of any errors.

With this installment, we have switched to listing entries alphabetically within CTAN directories, rather than by date. This seemed more useful to us, since it groups related entries together. Comments are welcome, as always.

Hopefully this column and those which follow will help to make CTAN a more accessible resource to the TEX community.

biblio

`apacite` in `biblio/bibtex/contrib`
: Attempts to implement the citation rules of the American Psychological Association.

`babelbib` in `biblio/bibtex/contrib`
: Generates multilingual bibliographies in cooperation with `babel`.

`bib-fr` in `biblio/bibtex/contrib`
: French translation of classical BibTEX styles.

`bibtool` in `biblio/bibtex/utils`
: Manipulation of BibTEX files including: sorting and merging, pretty-printing, syntax checks with error recovery, semantic checks, generation of uniform reference keys, controlled rewriting with regular expressions, collection of statistics, and more. Includes documentation. C source only (no binary).

`cj.bst` in `biblio/bibtex/contrib`
: A BibTEX file for *The Computer Journal* published by the British Computer Society.

`development` in `biblio/bibtex/contrib`
: BibTEX style for the journal *Development* (`http://dev.biologists.org/`).

`directory` in `biblio/bibtex/contrib`
: Facilitates the construction, the maintenance and the exploitation of an address book database. Version 1.18 introduces two new BibTEX styles that permit export of address books in vCard and LDIF formats (accepted by Apple's Address Book, Microsoft Outlook, Mozilla, etc.) without requiring any external programs.

`germbib` in `biblio/bibtex/contrib`
: Macros for German BibTEXing.

`gost` in `biblio/bibtex/contrib`
: BibTEX styles to format a bibliography in English, Russian, and Ukrainian according to GOST 7.1-84 and GOST 7.80-00.

`numalg` in `biblio/bibtex/contrib`
: BibTEX style for the journal *Numerical Algorithms*.

`urlbst` in `biblio/bibtex/contrib`
: Adds support for 'url', 'lastchecked' and 'eprint' fields to BibTeX style files.

`xbibfile` in `biblio/bibtex/utils`
: A graphical tool for creating and searching BibTeX databases. Written in C under Linux using the X Window System and GTK graphics libraries.

dviware

`BSR2dvi` in `dviware`
: Convert Textures files to standard `.dvi`.

`dvi2bitmap` in `dviware`
: Convert `.dvi` directly to bitmaps.

`dvipng` in `dviware`
: Convert `.dvi` to `.png` images.

fonts

`ae` in `fonts`
: A set of virtual fonts that emulate T1 encoded fonts with CM fonts.

`allrunes` in `fonts`
: This collection of fonts claims to give access to almost all runes ever used in Europe.

`antiqua` in `fonts/urw`
: URW Antiqua Condensed.

`astro` in `fonts`
: ASTROSYM is a font containing astronomical symbols, including those used for the planets, four planetoids, the phases of the moon, the signs of the zodiac, and some additional symbols.

`base35` in `fonts/urw`
: These are URW's PostScript base fonts, which are provided under the GPL and usually distributed in conjunction with Ghostscript. As far as TeX is concerned, pdfTeX, Dvips, and other applications need them as drop-in replacements for Adobe's non-free original base fonts.

`bookhands` in `fonts`
: METAFONT fonts and packages covering manuscript scripts from the 1st century until Gutenberg and Caxton. Includes Square Capitals, Insular Minuscule, Carolingian Minuscule, Early Gothic, Gothic Textura Quadrata, Gothic Textura Prescius, Rotunda, Humanist Minuscule, Roman Rustic, Uncial, Half Uncial, Artificial Uncial, and Insular Majuscule. The file `bsamples.ps` shows examples of each of the fonts.

`brokent1` in `fonts`
: Provides virtual fonts for T1-like variants of the broken yfrak, yswab, and ygoth typefaces published in 1990 by Yannis Haralambous. The structure of this package allows for broken typefaces from other sources to be made usable for LaTeX.

`cb` in `fonts/greek`
: Claudio Beccari's massive Greek font collection.

`cm-lgc` in `fonts/ps-type1`
: Type 1 fonts converted from METAFONT sources of the Computer Modern font families.

`cmtiup` in `fonts/cm`
: Unslanted punctuation in Computer Modern italic.

`corelfonts` in `fonts/utilities`
: Perl script to install all Bitstream fonts on a Corel Ventura CD for use with LaTeX.

`covfonts` in `fonts`
: Makes Apostrophic Laboratorie's Covington fonts available to TeX and LaTeX.

`DayRoman` in `fonts`
: A digitally redrawn version of what has come to be historically known as the "Two Line Double Pica Roman."

`doublestroke` in `fonts`
: Font for typesetting the mathematical symbols for natural numbers, real numbers, etc.

`eulervm` in `fonts`
: This is a set of *virtual* math fonts, based on Euler and CM. Included is a LaTeX package, which makes them easy to use, particularly in conjunction with Type 1 PostScript text fonts.

`euroitc` in `fonts`
: Provides a LaTeX interface to the PostScript euro font symbols which are available free of charge from the International Typeface Corporation.

`fontinst` in `fonts/utilities`
: A program that helps with installing fonts. Since it is written entirely in TeX macros, it is completely portable.

`fourier-GUT` in `fonts`
: Fourier-GUTenberg is a math complement for Adobe Utopia.

`frcursive` in `fonts`
: This is the French Cursive font, a cursive handwriting font family in the style of the French academic running-hand, written with METAFONT.

`glonti` in `fonts/cyrillic`
: Virtual fonts that combine CM and CMCYR.

`grotesq` in `fonts/urw`
: URW Grotesk Bold.

`hfbright` in `fonts/ps-type1`
: Contains the cmbright fonts in Type 1 format.

`hieroglf` in `fonts/archaic`
: Provides a METAFONT version of about 75 Egyptian hieroglyphs from Serge Rosmorduc's comprehensive `hieroglyph` package. Sufficient glyphs are provided for writing a few names like Cleopatra or Ptolomeny (together with cartouches) and some numbers. There is a command for transliterating into a modern alphabet. The package is not for serious Egyptologists.

`hieroglyph` in `fonts`
: HieroTeX is a package for typesetting ancient Egyptian hieroglyphs. It contains a hieroglyphic font, a

number of style files, and a helper program in C called sesh which allows one to type hieroglyphic texts using the so-called "manuel de codage", the current standard for encoding ancient Egyptian.

`i-ching` in `fonts/psfonts`
The I-Ching-Regular font, in Type 1 format, with macros and graphics for typesetting divinations.

`igo` in `fonts`
Fonts and macros to typeset Go diagrams.

`initials` in `fonts`
Several fonts for dropped initials.

`kerkis` in `fonts/greek`
Kerkis font family, based on URW Bookman, with complete Greek support.

`kixfont` in `fonts`
The postal services of a few countries use KIX to encode zip code + home number. This version is compatible with (at least) the Dutch system.

`leawood` in `fonts/psfonts`
Provides all the files needed to make the ITC Leawood font available to TEX and LATEX. The font itself is not freely available, so the package assumes you have already purchased ITC Leawood.

`lm` in `fonts/ps-type1`
The massive Latin Modern font collection, in Type 1 format.

`lucida` in `fonts/metrics/bh`
Updated font metrics and virtual fonts for Y&Y's commercial Lucida Bright fonts.

`LuxiMono` in `fonts`
From the designers of Lucida, a family of general-purpose monospaced (typewriter) fonts. May be freely copied but not modified.

`mbboard` in `fonts`
Blackboard bold fonts.

`mtype13` in `systems/unix`
METAPOST libraries, bash, and Perl scripts which help in programming and creation of Type 1 and Type 3 fonts under Linux.

`musixps` in `fonts/musixtex/ps-type1`
This package provides PostScript (Type 1) fonts (PFB format), and dvips and dvipdfm map files for MusiXTEX.

`psgreek` in `fonts/greek`
LATEX support for popular Type 1 Greek fonts in the WinGreek encoding.

`punk` in `fonts/punk/latex`
Updated style file and test LATEX file for Knuth's punk fonts.

`skak` in `fonts`
For typesetting chess games using the PGN standard.

`webomints` in `fonts`
A border and ornament font from Galapagos Design Group.

`viking` in `fonts/archaic`
The two 16-letter runic alphabets as used by the Vikings in Scandinavia.

`yfonts` in `fonts/ps-type1`
Yannis Haralambous' yfrak, yswab, and ygoth in Type 1 format.

`yhmath` in `fonts`
Provides big delimiters and very wide accents (including two new ones: parenthesis and triangle).

graphics

`3DLDF` in `graphics`
Three-dimensional (batch) drawing program with METAPOST output.

`bardiag` in `graphics`
Draw bar diagrams using PSTricks.

`bbcard` in `graphics/metapost/contrib/macros`
METAPOST examples for a "Bullshit Bingo" playing card, a macro which breaks text into paragraphs, a calendar, and a baseball score card.

`circuit_macros` in `graphics`
Macros for drawing high-quality electric circuit diagrams containing fundamental elements, amplifiers, transistors, and basic logic gates to include in TEX, LATEX, or similar documents. Tools and examples for other types of diagrams are also included.

`degrade` in `graphics`
Degrades JPEG images on the fly to decrease the size of the resulting PostScript or PDF file.

`epix` in `graphics`
Utility for mathematically accurate, camera quality plots and line figures.

`featpost` in `graphics/metapost/macros`
Three-dimensional drawing with METAPOST.

`finomaton` in `graphics`
Draw and typeset finite state machines (automata).

`gchords` in `graphics`
Typeset guitar chord diagrams, including options for chord names, finger numbers, and typesetting above lyrics.

`hatching` in `graphics/metapost/macros`
A set of METAPOST macros for hatching the interior of closed paths.

`jPicEdt` in `graphics`
A vector-based graphic editor for LATEX.

`latexmp` in `graphics/metapost/contrib/macros`
Interface for LATEX-based typesetting in METAPOST.

`makecirc` in `graphics/metapost/contrib/macros`
METAPOST library for electrical circuit diagrams.

`mfpic` in `graphics`
Macros which generate METAFONT or METAPOST for drawing pictures.

`mp3d` in `graphics/metapost/macros/3d`
Three-dimensional drawing with METAPOST.

`mpsproof` in `graphics/metapost/contrib/misc`
Produce proofs of METAPOST figures.

`pgf` in `graphics`
Updates to the TEX Portable Graphic Format.

`pst-3dplot` in `graphics/pstricks/contrib`
Plot 3D math functions.

`pst-circ` in `graphics/pstricks/contrib`
Draw electric circuits.

`pst-optic` in `graphics/pstricks/contrib`
Draw lenses and mirrors for optical systems.

`pst-uml` in `graphics/pstricks/contrib`
Draw UML diagrams.

`pst-vue3d` in `graphics/pstricks/contrib`
Draw 3D scenes.

`sam2p` in `graphics`
A UNIX command line utility that converts many raster (bitmap) image formats into Adobe PostScript or PDF files and other formats.

`shapepatch` in `graphics/transfig-shapepatch`
ShapePatch is a patch against `transfig` that adds a new output driver for the TEX macro `shapepar` by Donald Arsenau. Using this driver one can sketch the shape in XFig and then convert it to `shapepar`.

`texcad32` in `graphics`
TEXCad32 is a clone of the DOS program Texcad running under Windows.

`transfig` in `graphics`
A set of tools for creating TEX documents with graphics which are portable, in the sense that they can be printed in a wide variety of environments.

`xfig` in `graphics`
A menu-driven tool for drawing and manipulating objects interactively in an X window. The resulting pictures can be saved, printed on PostScript printers or converted to a variety of other formats (e.g., to allow inclusion in LATEX documents).

help

`es-tex-faq` in `help`
The CervanTEX FAQ from the Spanish TEX Users Group.

`uk-tex-faq` in `help`
Major English-language FAQ, with much information on virtually all TEX-related topics. Current version always available via the Web at `http://www.tex.ac.uk/faq`.

indexing

`authorindex` in `indexing`
Create an index of authors cited in a document.

info

`beginlatex` in `info`
Formatting Information: A Beginner's Guide to LATEX.

`comprehensive` in `info/symbols`
The Comprehensive LATEX Symbols List is an organized list of over 2500 symbols commonly available to LATEX users. Some of these symbols are guaranteed to be available in every TEX distribution. Others require font files that come with some, but not all, TEX distributions. The rest require font files that must be downloaded explicitly from CTAN and installed.

`fontinstallationguide.pdf` in `info/Type1fonts`
A guide to installing Type 1 PostScript fonts.

`fontsampler` in `info`
Samples of free typefaces available with LATEX.

`impatient` in `info`
TEX for the Impatient was originally published in 1990 by Addison-Wesley. It has tutorial and reference information on primitive and plain TEX; it does not discuss LATEX. It is now freely available.

`l2tabu` in `info`
Mark Trettin's guide to LATEX, Das LATEX 2ε-Sündenregister. Originally in German with translations available in English and Italian.

`LaTeX2PDF.pdf` in `info/german`
Erstellung von PDF-Dokumenten mit LATEX is a German guide to creating a PDF document with LATEX and hyperref/thumbpdf.

`latex4wp` in `info`
A guide for word processor users designed to help convert knowledge and techniques of word processing into the LATEX environment.

`lshort` in `info`
"The Not Short Introduction to LATEX 2ε". Originally in English, with translations in Dutch, Finnish, German, Portuguese, Russian, and Ukrainian.

`makingtexwork` in `info`
The O'Reilly book *Making TEX Work*, now freely available.

`MiKTeX-WinEdt-TrueType-Anleitung` in `info/german`
German information about MikTEX, WinEdt, and TrueType.

`ttf-tetex` in `info/TrueType`
The tutorial "Using TrueType fonts with teTEX and dvips" that describes how to use TrueType fonts with teTEX, by converting them to Type 1 PostScript fonts. It does not describe how to use TrueType fonts directly.

language

`bgreek` in `language/greek`
Typeset classical Greek.

`CJHebrew` in `language/hebrew`
Hebrew typesetting package including fonts.

`CJK` in `language/chinese`
(V.4.5.2) LATEX support for Asian scripts: Chinese (both traditional and simplified), Japanese, Korean and Thai, in many encodings (including Unicode).

`epiolmec` in `language`
Typeset Epi-Olmec, a script used in Southern Middle America until about 500 AD.

`eshyph` in `language/hyphenation`
Spanish hyphenation patterns.

`frenchle` in `language/french`
Installation changes for Unix and Mac OS X.

`hebclass` in `language/hebrew`
Hebrew LaTeX classes.

`ibycus-babel` in `language/greek/package-babel`
Allows usage of the Ibycus 4 font for ancient Greek with Babel.

`makor` in `language/hebrew`
Makor 2 typesets Hebrew with vowels or liturgical accents, Yiddish, and more.

`oinuit` in `language/inuktitut`
Typesetting tools for Inuktitut documents..

`omega` in `language/devanagari`
Typeset Devanagari texts with Omega.

`ruhyphen` in `language/hyphenation`
Updates to this package for Russian hyphenation.

`serto` in `language/aramaic`
Fonts, style files, and a preprocessor to typeset Syriac (Aramaic).

`srhyphc` in `language/hyphenation`
Serbian hyphenation patterns.

`velthuis` in `language/devanagari`
Velthuis Devanagari for TeX.

macros/context

`t-amsl` in `macros/context/contrib/maths`
Provides some environments and commands that AMS-LaTeX users expect.

`t-nath` in `macros/context/contrib/maths`
Provides for ConTeXt the same functionality as the `nath` package for LaTeX.

macros/generic

`dcpic` in `macros/generic/diagrams`
Macros for drawing commutative diagrams in a TeX (including LaTeX and ConTeXt) document.

`longdiv` in `macros/generic/misc`
Work out and print integer long division problems.

`petri-nets` in `macros/generic`
Draws Petri nets and related models.

`scripttex` in `macros`
Format screenplays and other scripts.

macros/latex

`TeXPower` in `macros/latex/exptl/texpower`
A bundle of LaTeX packages and classes for making dynamic online presentations.

macros/latex/contrib

`acronym` in `macros/latex/contrib`
Ensures that all acronyms used in the text are spelled out in full at least once. Also provides an environment to keep a list of used acronyms.

`algorithm2e` in `macros/latex/contrib`
An environment for writing algorithms.

`algorithmicx` in `macros/latex/contrib`
Include pseudocode or source code in papers.

`alnumsec` in `macros/latex/contrib`
Alphanumeric sectioning numbering with standard sectioning commands.

`apa` in `macros/latex/contrib`
Typeset documents according to the APA manual (5th ed).

`appendix` in `macros/latex/contrib`
Provides various ways for formatting the titles of appendices.

`beamer` in `macros/latex/contrib`
Create slides and presentations with a projector.

`begriff` in `macros/latex/contrib`
Typeset Frege's Begriffsschrift.

`betababel` in `macros/latex/contrib`
Insert ancient Greek text coded in Beta Code into your document.

`bgteubner` in `macros/latex/contrib`
Class for books published by Teubner Verlag.

`bibcheck` in `macros/latex/contrib/misc`
Checks that every entry in a `thebibliography` environment has been cited.

`bibtopic` in `macros/latex/contrib`
Include several bibliographies covering different 'topics' into a document.

`bitfield` in `macros/latex/contrib`
Draws bitfield diagrams.

`blindtext` in `macros/latex/contrib`
Create 'greeking' text for testing documents.

`booklet` in `macros/latex/contrib`
Provides some aids for printing simple booklets or signatures for longer books.

`booktabs` in `macros/latex/contrib`
Enhance the quality of tables in LaTeX.

`bpchem` in `macros/latex/contrib`
Package for chemical typesetting.

`bytefield` in `macros/latex/contrib`
Create illustrations for network protocol specifications.

`caption` in `macros/latex/contrib`
Provides an easy-to-use interface to customise the layout of figure and table captions.

`carlisle` in `macros/latex/contrib`
A collection of packages by David Carlisle, including: `colortbl` (add colour to tables), `fix2col` (keep `\firstmark` and float order in `twocolumn`), `nopageno`

(turn off page numbers), `scalefnt` (scale fonts relative to current size), `tabulary` (different column width allocation algorithm).

`ccaption` in `macros/latex/contrib`
Provides continuation captions, unnumbered captions and legends.

`chngpage` in `macros/latex/contrib/misc`
Provides commands to change the page layout in the middle of a document and to robustly check for typesetting on odd or even pages.

`cite` in `macros/latex/contrib`
Compressed, sorted lists of numerical citations.

`clrscode` in `macros/latex/contrib`
Typeset pseudocode in the style of *Introduction to Algorithms*.

`cmap` in `macros/latex/contrib`
Create searchable PDF files.

`colorinfo` in `macros/latex/contrib`
Retrieve color model and color values for already defined colors.

`combine` in `macros/latex/contrib`
Bundle individual documents into a single document, e.g., a conference proceedings.

`comicsans` in `macros/latex/contrib`
Use Microsoft's Comic Sans font with LaTeX.

`contour` in `macros/latex/contrib`
Generates a colored contour around a given text in order to enable printing text over a background without the need for a color box around the text.

`crop` in `macros/latex/contrib`
Provides different forms of cropmarks for trimming paper stacks, for camera alignment and for visualizing the page dimensions.

`csquotes` in `macros/latex/contrib`
Markup for inline quotations in terms of control sequences or active quote characters.

`ctable` in `macros/latex/contrib`
Typeset centered table and figure floats with footnotes.

`curve` in `macros/latex/contrib`
A class for making curriculum vitæ .

`custom-bib` in `macros/latex/contrib`
Customize BibTeX style files.

`datetime` in `macros/latex/contrib`
Commands to print the current time (12 or 24 hour forms), ordinal number forms (e.g., 3rd), and as a string (e.g., `\numberstring{3}` would print three).

`dblfloatfix` in `macros/latex/contrib`
Fixes for `twocolumn` floats.

`decimal` in `macros/latex/contrib`
Use the traditional English decimal point instead of the American-style period.

`dnaseq` in `macros/latex/contrib`
Typeset simple single stranded DNA sequences (or any other letter sequence) in fixed-length blocks. The number of blocks per line is computed automatically, and the lines are prefixed with the current sequence number.

`dramatist` in `macros/latex/contrib`
Typeset dramatic works (plays).

`ebezier` in `macros/latex/contrib`
Plot Bézier curves in the picture environment.

`eCards` in `macros/latex/contrib`
Create electronic flash cards.

`ednotes` in `macros/latex/contrib`
Critical edition typesetting with LaTeX.

`eemeir` in `macros/latex/contrib`
Facilitates writing documents that must be produced in both male and female forms by providing natural commands to type in place of gender specific words.

`ellipsis` in `macros/latex/contrib`
Fixes the uneven spacing around ellipses in text mode.

`empheq` in `macros/latex/contrib`
A visual markup extension to `amsmath` for emphasizing equations.

`engpron` in `macros/latex/contrib`
Typeset English pronunciation.

`epigraph` in `macros/latex/contrib`
Typeset epigraphs, the pithy quotations often found at the start (or end) of a chapter.

`eqlist` in `macros/latex/contrib`
Allows description-like lists to have equal indentation. Requires `eqparbox`.

`eso-pic` in `macros/latex/contrib`
Makes it easy to add some picture commands to every page.

`evautofl` in `macros/latex/contrib/calendar/contrib`
An extension of the `autofilo.cls` of the LaTeX Calendar Bundle. The code has been hacked to separate the two columns making the page in `autofilo`, so now it's possible to use `ps2ps` to put more than a single page on a sheet when printing your calendar, thus saving a lot of paper.

`evweek` in `macros/latex/contrib/calendar/contrib`
An extension of the `weekly.cls` of the LaTeX Calendar Bundle.

`excludeonly` in `macros/latex/contrib/misc`
Define `\excludeonly`, the opposite of `\includeonly`.

`facsimile` in `macros/latex/contrib`
Create faxes.

`fbithesis` in `macros/latex/contrib`
At the University of Dortmund there are cardboard cover pages for research or internal reports. The main function of this document class is to typeset a title page that is adjusted to these cover pages.

`fnbreak` in `macros/latex/contrib`
Writes a warning to the log file when footnotes are split over several pages.

`fncylab` in `macros/latex/contrib/misc`
Changes the way labels are defined.

`footbib` in `macros/latex/contrib`
Make bibliographic references appear as footnotes.

`footmisc` in `macros/latex/contrib`
Customize footnotes.

`framed` in `macros/latex/contrib/misc`
Creates framed or shaded regions that can break across pages.

`g-brief` in `macros/latex/contrib`
Format formless letters in German or English.

`gatech-thesis` in `macros/latex/contrib`
Georgia Institute of Technology theses.

`gauss` in `macros/latex/contrib`
Package for typesetting matrix operations,

`gensymb` in `macros/latex/contrib/was`
Provides generic commands `\degree`, `\micro`, `\ohm`, `\celsius` and `\perthousand`, which work both in text and math mode. Various means are provided to fake the symbols or take them from particular symbol fonts, if they are not available in the default fonts used in the document.

`gloss` in `macros/latex/contrib`
Allows creation of glossaries via BibTeX.

`guit` in `macros/latex/contrib`
Provides commands to correctly write the logo of "Gruppo Utilizzatori Italiani di TeX" (Italian TeX User Group).

`hhtensor` in `macros/latex/contrib`
Commands for vectors, matrices and tensors.

`hvfloat` in `macros/latex/contrib`
Rotating float objects and captions.

`iagproc` in `macros/latex/contrib/misc`
LaTeX 2_ε class file for two column *IAG Proceedings* articles.

`interactiveworkbook` in `macros/latex/contrib`
Create interactive question-and-answer PDF tutorials meant to be used by Internet students.

`invoice` in `macros/latex/contrib`
For writing invoices. Supports English, German, Dutch and French.

`iso` in `macros/latex/contrib`
Typeset ISO International Standard documents.

`isodate` in `macros/latex/contrib`
Print dates in a variety of formats.

`jurabib` in `macros/latex/contrib`
Enables automated citation with BibTeX for legal studies and the humanities.

`keystroke` in `macros/latex/contrib`
For typesetting the graphical representation of the keys on a computer keyboard.

`koma-script` in `macros/latex/contrib`
A large and notable replacement for the standard LaTeX 2_ε classes.

`ktv-texdata` in `macros/latex/contrib`
Manage libraries of mathematics exercises. Useful for teachers.

`labbook` in `macros/latex/contrib`
Typeset laboratory journals that contain chronologically ordered records about experiments.

`labels` in `macros/latex/contrib`
Make sticky labels.

`layouts` in `macros/latex/contrib`
Enables the visual display of various elements of a document's layout.

`ledmac` in `macros/latex/contrib`
Typeset critical editions; a LaTeX equivalent of the plain `edmac` macros.

`leftidx` in `macros/latex/contrib`
Enables left subscripts and superscripts in math mode.

`lettre` in `macros/latex/contrib`
Write letters and faxes in French, English, and German.

`lettrine` in `macros/latex/contrib`
Package designed to typeset various sorts of dropped capitals.

`lineno` in `macros/latex/contrib`
Provides line numbers on paragraphs.

`linsys` in `macros/latex/contrib/misc`
Inserts a circled number to the left of each equation in a linear system.

`ltabptch` in `macros/latex/contrib`
Fixes bugs in `longtable.sty`.

`manuscript` in `macros/latex/contrib`
Emulates the look of a document typed on a typewriter.

`mceinleger` in `macros/latex/contrib`
Typeset cassette covers.

`memoir` in `macros/latex/contrib`
Peter Wilson's flexible LaTeX `documentclass` for typesetting of books such as novels, biographies, histories, etc., with options for trim marks, draft appearance, various sizes and much more.

`menu` in `macros/latex/contrib`
Typeset GUI menu selections for software documentation.

`miller` in `macros/latex/contrib`
Typesets Miller indices as used in material science, where negative numbers are written with a bar over them.

`minitoc` in `macros/latex/contrib`
Create mini-tables of contents by chapter, by section, or by parts.

`mla-paper` in `macros/latex/contrib`
Typeset papers in the MLA style.

`modroman` in `macros/latex/contrib/misc`
Write lower case roman numerals.

`mparhack` in `macros/latex/contrib`
Ensure that marginpars appear on the correct margin.

`mtpro` in `macros/latex/contrib`
Support for the commercial MathTimeProfessional fonts with LaTeX.

`multibib` in `macros/latex/contrib`
Create references to multiple bibliographies within one document.

`mwcls` in `macros/latex/contrib`
A set of document classes for LaTeX 2ε designed with the Polish typographical tradition in mind.

`namespc` in `macros/latex/contrib`
Rudimentary C++-like namespaces in LaTeX.

`nath` in `macros/latex/contrib`
Natural math notation, a style to separate presentation and content in mathematical typography. Delivers a particular context-dependent presentation on the basis of a rather coarse context-independent notation — aims for producing traditional math typography output.

`ncctools` in `macros/latex/contrib`
LaTeX 2ε packages written and supported by Alexander I. Rozhenko.

`needspace` in `macros/latex/contrib/misc`
Provides commands to reserve space at the bottom of a page.

`newlfm` in `macros/latex/contrib`
For creating letters, faxes and memos; integrates the `letter` class with `fancyhdr` and `geometry` and includes support for an address database, languages, Avery labels and has full documentation.

`nolbreaks` in `macros/latex/contrib/misc`
Attempts to prevent line-breaks in portions of text while still allowing flexible glue.

`notes` in `macros/latex/contrib`
A style for highlighting notable sections of text in a document by putting the text in a boxed frame and placing a small graphic in the margin. Specifically designed to work with double sided pages, placing the 'icon' in the correct margin.

`numprint` in `macros/latex/contrib`
Pretty printing numbers.

`octavo` in `macros/latex/contrib`
Typeset books following classical layout and design principles.

`opcit` in `macros/latex/contrib`
Footnote-style bibliographical references.

`outline` in `macros/latex/contrib`
A six-level list environment for making outlines.

`parallel` in `macros/latex/contrib`
Typeset in two columns or two pages in parallel, e.g. typeset two languages side-by-side.

`parrun` in `macros/latex/contrib`
Typeset two streams of text running parallel.

`pbox` in `macros/latex/contrib`
A variable-width `\parbox`.

`pdfpages` in `macros/latex/contrib`
Insert pages of external PDF documents.

`perpage` in `macros/latex/contrib/misc`
Adds the ability to reset counters per page.

`perltex` in `macros/latex/contrib`
Combines LaTeX's typesetting power with Perl's programmability.

`platex` in `macros/latex/contrib`
Provides tools to typeset documents in Polish using LaTeX 2ε.

`poemscol` in `macros/latex/contrib`
Typeset critical editions of poetry.

`polytable` in `macros/latex/contrib`
`tabular`-like environments with named columns.

`ppr-prv` in `macros/latex/contrib`
Produce a printable version of slides written with Prosper, with two slides per page.

`proba` in `macros/latex/contrib`
Shortcut commands for symbols used in probability texts.

`progress` in `macros/latex/contrib`
Creates an overview of a document's state.

`prosper` in `macros/latex/contrib`
Slides using LaTeX.

`ps4pdf` in `macros/latex/contrib`
Use PostScript code in pdfLaTeX documents.

`psfragx` in `macros/latex/contrib`
Embed `\psfrag` commands provided by the `psfrag` package into the EPS file itself.

`ragged2e` in `macros/latex/contrib/ms`
Defines new commands which set ragged text and are easily configurable to allow hyphenation.

`refstyle` in `macros/latex/contrib`
Advanced formatting of cross-references.

`register` in `macros/latex/contrib`
Typesets the programmable elements in digital hardware, i.e., registers.

`relsize` in `macros/latex/contrib/misc`
Set the font size relative to the current size.

`rotpages` in `macros/latex/contrib`
Typeset multiple pages upside-down with page order rearrangement.

`sansmath` in `macros/latex/contrib/misc`
Define a new math version 'sans' and a command '`\sansmath`', much like '`\boldmath`'.

`savesym` in `macros/latex/contrib`
Saves and restores symbols.

`scalebar` in `macros/latex/contrib`
Creates scalebars for maps, diagrams, or photos.

`SciWordConv` in `macros/latex/contrib`
Use Scientific Word and Scientific WorkPlace source files with another TeX compiler.

`semantic` in `macros/latex/contrib`
Help for writing programming language semantics, including software ligatures.

`sffms` in `macros/latex/contrib`
Typeset fiction manuscripts.

`shadow` in `macros/latex/contrib/misc`
Draw a box with a drop shadow.

`shapepar` in `macros/latex/contrib/misc`
Typeset paragraphs of a specified shape, where the total size is adjusted automatically so that the entire shape is filled with text, and the shape may include separate pieces and holes.

`sidecap` in `macros/latex/contrib`
Typeset captions sideways.

`SIunits` in `macros/latex/contrib`
Typeset physical units following the rules of the International System of Units (SI).

`slantsc` in `macros/latex/contrib`
Enables small capitals in different font shapes.

`smalltableof` in `macros/latex/contrib`
Moves List of Figures or List of Tables to the sections of a chapter.

`soul` in `macros/latex/contrib`
Provides flexible, hyphenatable letterspacing, underlining, overstriking, and highlighting.

`SplitIndex` in `macros/latex/contrib`
Use an unlimited number of indices.

`sseq` in `macros/latex/contrib`
Draw spectral sequence charts.

`statex` in `macros/latex/contrib`
A statistics style for LaTeX.

`statistik` in `macros/latex/contrib`
Writes the page numbers of each chapter (plus chapter number and title) into an external file. Several LaTeX, CSV and XML formats can be produced.

`subfloat` in `macros/latex/contrib`
Enables subnumbering of different floats.

`svn` in `macros/latex/contrib`
Typeset Subversion keywords.

`svninfo` in `macros/latex/contrib`
Extracts the revision and file information provided by the Subversion revision control system.

`tabvar` in `macros/latex/contrib`
Eases the typesetting of tables showing variations of functions as they are used in France. The documentation is in French.

`tclldoc` in `macros/latex/contrib`
Simplify literate programming with Tcl.

`teubner` in `macros/latex/contrib`
A complement to the `greek` option of the `babel` package supporting "Lipsian" fonts, similar to those used by the Teubner Printing Company of Lipsia.

`texlogos` in `macros/latex/contrib`
Defines a number of ready-to-use LaTeX logos.

`textpos` in `macros/latex/contrib`
Fixes spacing misfeatures.

`threeparttable` in `macros/latex/contrib`
Tables with titles (captions) and notes. The titles and notes are given a width equal to the body of the table.

`thrmappendix` in `macros/latex/contrib/misc`
Facilitates moving long proofs to an appendix.

`tocbibind` in `macros/latex/contrib`
Add the Table of Contents, Bibliography, or Index to the Table of Contents listing.

`tocloft` in `macros/latex/contrib`
Provides control over the typography of the Table of Contents, List of Figures and List of Tables.

`todo` in `macros/latex/contrib`
Append a "to-do" list to a document.

`tokenizer` in `macros/latex/contrib`
Tokenizes comma-separated lists of strings.

`toolbox` in `macros/latex/contrib`
Provides some macros which are convenient for writing indices, glossaries, or other macros.

`tugboat` in `macros/latex/contrib`
LaTeX classes for writing *TUGboat* articles.

`typogrid` in `macros/latex/contrib`
Produces a typographic grid on every page of the document.

`upgreek` in `macros/latex/contrib/was`
Makes the Euler or Symbol typeface available as an upright Greek math alphabet.

`upquote` in `macros/latex/contrib`
Switches the typewriter font to Computer Modern Typewriter for its upright single quotes.

`url` in `macros/latex/contrib/misc`
Typeset and allow line breaking in URLs.

`uwmslide` in `macros/latex/contrib`
University of Wisconsin-Madison slides. Produces slides with a simple PowerPoint like appearance. Several different slide environments are included.

`varwidth` in `macros/latex/contrib/misc`
Variable width minipages.

`verse` in `macros/latex/contrib`
Aids in typesetting simple verse (poems).

`versions` in `macros/latex/contrib`
Optionally omit pieces of input.

`wasysym` in `macros/latex/contrib`
Defines commands to make some additional characters available from the `wasy` fonts.

`webeq` in `macros/latex/contrib`
The AcroTeX eDucational Bundle is a series of packages designed primarily for online education. Design your online page layout; create online exercises and quizzes; add Acrobat eForm elements; add Acrobat JavaScript to make your document interactive; submit your online quizzes to a server-side script (some basic scripts included).

`withesis` in `macros/latex/contrib`
University of Wisonsin-Madison theses.

`wrapfig` in `macros/latex/contrib`
Places a figure or table at the side of the page and wrap text around it. Includes a recipe for multiple column inserts.

`xcolor` in `macros/latex/contrib`
Driver-independent color extensions.

`xdoc` in `macros/latex/contrib`
Enhanced rewrite of the `doc` package.

`yfonts` in `macros/latex/contrib`
Support for the blackletter (old-German) typefaces provided by Y. Haralambous.

macros/plain

`figflow` in `macros/plain/contrib`
: Plain TEX macro to flow text around a figure.

`metatex` in `macros/plain/contrib`
: Provides two way communication between TEX and METAFONT to allow both the text and the figures in a single source file.

`tugboat` in `macros/plain/contrib`
: Plain TEX macros for writing *TUGboat* articles.

nonfree

`FoilTeX` in `nonfree/macros/latex/contrib/supported`
: A system for generating transparencies and slides.

`initials` in `nonfree/fonts`
: A collection of fonts for decorative initials.

`oriya` in `nonfree/language`
: Typesetting in Oriya, an Indian script.

`rst` in `nonfree/macros/latex/contrib/supported`
: Draw rhetorical structure analysis diagrams.

support

`bibtex-gen` in `support`
: A simple interactive script to generate BibTEX files.

`bmeps` in `support`
: A program to convert from PNG, TIFF, JPEG, and NetPBM to EPS.

`chktex` in `support`
: Finds typographic errors in LATEX.

`eps2pdf` in `support`
: A GUI interface for conversion of EPS files into PDF easier via Ghostscript.

`eukleides` in `support`
: A Euclidean geometry drawing language.

`fi2t1` in `support/miktex-contrib`
: Tools for installing Type 1 fonts in MikTEX.

`highlight` in `support`
: Command line tool to convert source code to syntax-highlighted (LA)TEX.

`isi2bib-vim` in `support`
: Vim script to convert a bibliographic database from ISI to BibTEX format.

`ite` in `support`
: An interactive tool for authoring LATEX and TEX documents within Emacs.

`JS-TeX` in `support/javascript_TeXed`
: Small MikTEX editor for Win32 Intel platform. Requires IE 5.0+.

`latexcount` in `support/misc`
: Generates word count of LATEX documents.

`latexdb` in `support`
: Brings together LATEX and a MySQL database. You can use SQL queries in your LATEX document and loop over the result sets creating tables, serial letters, and so on.

`latexjed` in `support`
: A LATEX mode for the Jed editor.

`lilypond` in `support`
: The GNU Project music typesetter, LilyPond, produces beautiful sheet music.

`mimetex` in `support`
: Parses well-formed LATEX math expressions, emitting either GIF images or MIME xbitmaps.

`mk` in `support/latex_maker`
: A Perl script that, in close collaboration with `vpp`, is helpful in the cyclic process of editing, viewing, and printing a LATEX document. `mk` uses the `make` utility for the management of file dependencies, `texi2pdf` for compilation, `gv` (or any other viewer you like) for viewing, and ConTEXt's `texexec` for printing in various formats.

`mps2eps` in `support`
: Convert a METAPOST output file (`.mps`) to a self-contained EPS file.

`pdfcrop` in `support`
: Takes a PDF as input, calculates the BoundingBox for each page with the help of Ghostscript and generates an output PDF without margins.

`png2pdf` in `support`
: Convert PNG images to PDF.

`preview-latex` in `support`
: A system for displaying inline images of selected parts of a file in Emacs source buffers. The style file is independently useful for extraction of selected text elements as images.

`proof` in `support`
: A Bash (Bourne Again Shell) based complement to an ordinary word processing program, able to coordinate the processing and viewing of TEX, LATEX, METAFONT and METAPOST-sources..

`ps2eps` in `support`
: Convert PostScript to EPS.

`psfragger` in `support`
: A free tool used to replace labels in EPS files by using `psfrag` and LATEX. The result is a modified EPS file that can be further converted to PDF for use with PDFLATEX (EPS to PDF conversion is included in this tool).

`references` in `support`
: Bibliographic software for authors of scientific manuscripts and for management of bibliographic data of journal articles, books, book chapters, etc.

`shlatex` in `support`
: LATEX compilation Bash script for Linux.

`tex4ht` in `support`
: A complete system for translating (LA)TEX sources into hypertext.

`texpack` in `support`
: A bundle of scripts to create documented LATEX styles, class files, and documentation.

`texpert` in `support`
: A German language TEX editor for Windows.

`tif2eps` in `support/pstools`
Convert TIFF images to EPS.

`ttf2tex` in `support`
Bash script to create all files necessary to use TrueType fonts with teTEX.

`txt2tex` in `support`
Convert plain text into LATEX.

`vpp` in `support/view_print_ps_pdf`
A command line utility to view and print PostScript and PDF documents.

`wp2latex` in `support`
Convert WordPerfect to LATEX.

systems

`bakoma` in `systems/win32`
Upgrade of BaKoMa TEX system to V.6.15.

`clasen` in `systems/tex-extensions`
Proposals for extensions to TEX by Matthias Clasen.

`enctex` in `systems`
A TEX extension supporting flexible input/output reencoding, with full UTF-8 (Unicode) processing.

`eomega` in `systems`
A TEX extension with both Omega's and e-TEX's enhanced features. The package will be renamed Aleph at the point of the first stable release.

`epmtfe` in `systems/os2`
The "EPM TEX Front End", a module for the OS/2 "Enhanced Editor" EPM. It turns the EPM into an integrated TEX environment, providing (LA)TEXing, previewing and executing of auxiliary programs from the editor menu.

`latexpix` in `systems/win32`
A drawing program for Windows which generates LATEX pictures.

`pdftex` in `systems`
An extension of TEX that can create PDF directly from TEX source files. It also supports new microtypographic features.

`WinShell` in `systems/win32`
A graphical user interface for easily working with TEX. It is *not* a TEX system itself, so requires a system such as MikTEX or TEX Live.

`wintex2000` in `systems/win32`
A shareware Windows TEX editor with Microsoft Office look and feel.

◇ Mark LaPlante
109 Turnbrook Drive
Huntsville, AL 35824
laplante@mac.com

Book Reviews

Book review: *The LATEX Companion, Second Edition*

Claudio Beccari

Frank Mittelbach, Michel Goossens, with Johannes Braams, David Carlisle, and Chris Rowley, with the contributions of Christine Detig and Joachim Schrod, *The LATEX Companion, Second Edition*. Addison-Wesley 2004, pp. xxviii+1092, ISBN 0-201-36299-6, USD 59.99, CND 86.99, ≈EUR 50.00.

The second edition of the indispensable *LATEX Companion* upgrades the first edition published approximately ten years ago. But "upgrades" is an understatement: the second edition is about two times as large as the first one, and two times as many pages implies that the material in this new edition contains much more than just a few references to new packages or a couple of extra examples.

Although the topics covered are essentially the same as those of the first edition, the fourteen chapters, three appendices, and indices are completely rewritten and rich with displayed and numbered examples that, with a clever programming decision, are set up in such a way as to be completely faithful to the material being shown. The CD-ROM attached to the book contains a slightly reduced version of the TEX Live distribution with the full running collection of the displayed examples, so that every reader can check directly also the details that are omitted from the typed page.

The chapters, after a good introduction, cover in order:

- the structure of a LATEX document,
- the basic formatting tools,
- how to specify the layout of the page,
- how to typeset tabular material in a professional way,
- how to master floats,
- a clear discussion on fonts and encodings,
- how to typeset higher mathematics,
- how to use LATEX in a multilingual environment,
- how to produce and handle graphical material,
- the organization and the tools for the difficult task of generating one or more indices,
- how to manage citations and produce useful bibliographies with the powerful tools that come with any distribution of (LA)TEX,

- and lastly, the tools for documenting class and package files.

The appendices start with a large overview on the commands usable in the document preamble and the more elaborate setups for package and class files. They continue with a very good analysis of the TEX, LATEX, and class or package messages that show up when some error or other problems take place; this appendix is precious and most of its material has never been published before. The last appendix refers itself to the CTAN archives, where most extension packages reside, and to the various unusual sites from which to fetch the other files.

I do not want to go into the details of every chapter or appendix; it would be too lengthy. At the same time, whoever has used the first edition of the *LATEX Companion* knows very well that this kind of book must be used as a descriptive manual; for this purpose the authors have written every chapter or appendix to be as "standalone" as possible, with an abundance of cross references allowing the readers to more deeply explore particular topics.

I would like to comment on a couple of points that I found a little weak or not sufficiently treated. This is not meant to criticize such an excellent and indispensable book as this one, but for contributing a line of thought that may help the readers (as well as the authors if they write a third edition in another ten years ...).

One point is the presence of "typos", that is spelling errors; actually the book is virtually devoid of any real typo, but there are some "cut and paste" errors, an average of two or three per chapter. This shows the amount of attention and accuracy the authors dedicated to correct and revise their source files; I have never succeeded in writing my own books with such a small number of spelling and/or "cut and paste" errors, so I admire them and praise their skill. But this raises another point: how is it possible to match such a beautiful typesetting engine as LATEX with a suitable editor that is aware of "cut and paste" errors? I know (or imagine) the authors used emacs, the most powerful ASCII editor available to anyone willing to climb its steep learning curve, but even this powerful tool is not capable of giving even a modest warning in such instances. I know that spell checking is one thing and grammatical checking is another, totally different. But at the end one would like to produce beautiful books, as TEX was designed for, and one cannot avoid the painful task of reading over and over the galley proofs, and at the end it becomes such a tiring activity that errors sneak in anyway or remain undetected.

The few errors that bothered me a little bit are those that appear in Appendix B, where the treatment of errors should be absolutely error free, otherwise a reader can't understand where is the error.

In Appendix B a slight confusion arises concerning `\long` macro definitions; for example in the unnumbered example starting at the end of page 932 there is an apparent inconsistency where `\lvec` is defined with `\newcommand`, that generates "long" macros, in contrast with the `\show\lvec` command that displays a "short" macro. The explanation of this unusual behavior follows immediately after the `\show\lvec` command output, but when the reader reaches that point it remains a puzzle from the output of the preceding unnumbered example on the same page, where the macro `\xvec` (defined without optional arguments) appears to be "long".

A naive error dealing with the Greek fonts appears in table 9.10: capital Greek letters are never accented, except for the diaeresis on Iota and Upsilon; a capital initial of a lowercase word is preceded by its spirit and accent, so that ᾽Ã, for example, is nonsense in Greek; it should be ᾼA as a capital initial, or simply A within an all caps word.[1] A similar situation holds true for the diaeresis: iota and upsilon receive the diaeresis only when they *follow* another vowel with which they are not supposed to form a diphthong so that ῞Υ is nonsense, because the capital upsilon preceded by its diacritical signs plays the role of an initial capital and therefore it cannot be part of a would-be diphthong; in the middle of a word it should simply be Ϋ.

In chapter 9 it would have been useful to cite the book on LATEX written by Apolostolos Syropoulos [2]; since the author is Greek he devoted a large part of his book to typesetting in languages different from English, and with non-Latin alphabets; he even illustrated the Mongolian script...

Another point I think is not sufficiently emphasized is the fulfillment of international standards, especially the ISO ones. I believe that international standards exist explicitly for helping people from all over the world to understand each other, at least on technical matters. I saw in the book the ISO standards mentioned in connection to the font encodings, and that's good. But I did not see a word about the ISO standard where, for example, table 9.5 illustrates the alternative mathematical operators names for eastern European languages. Those names are specified by the ISO standards and any alternative

[1] Actually the authors do not specify which Greek font they used, but with the default cbgreek fonts it is quite difficult to produce the errors shown in table 9.10; in this text I had to cheat a little bit in order to reproduce those errors!

name is to be considered "illegal"; I understand the necessity of producing documents containing some text typeset in accordance with obsolete typesetting traditions, or books containing translations of the ISO (Latin abbreviations of the) mathematical operators, as Apostolos Syropoulos did for math documents devoted to young high school Greek students. But in such an eminent reference as the *LaTeX Companion* I'd have put a discouraging sentence for common everyday use of nonstandard names. I know the LaTeX special symbols, now replaced by the American Mathematical Society or Text companion fonts, contained the ʊ symbol; maybe when LaTeX was first introduced in 1985 in some countries that symbol was in common use, but the ISO standards deprecate it explicitly and now I rarely see it in new books; even in this case a discouraging sentence recalling the international rules would have been appreciated.

This topic could be extended also to the bibliographies, for example, where emphasis is given to the many tools for typesetting bibliographic references, with the aid of BibTeX, and with the purpose of producing the fanciest citation schemes, without mentioning any of the ISO standards on the matter.

There is a third point I would like to comment on, namely the excessively discouraging sentences connected with the use of primitive TeX commands. I agree that when one writes a new class or a new package the use of the powerful LaTeX commands should be preferred over the often obscure circumlocutions needed when using primitive TeX commands, but there are some things that even the most powerful LaTeX commands cannot do; examples are the definition of macros with delimited arguments, and the conditional statements dealing with character and category codes, as well as those dealing with comparisons of control sequences.

For the former it is necessary to use the `\def` or the `\gdef` primitives, and for the latter it is necessary to use the `\if`, `\ifcat`, and `\ifx` primitive conditional commands. The whole LaTeX kernel contains such commands, some of them relics of the old LaTeX 209 kernel, but most of them are there simply because they are necessary and cannot be substituted with newer LaTeX commands (which, in any case, are eventually defined by means of those TeX primitives). Of course it is dangerous to make definitions by means of the primitive commands, because there is the real danger of redefining essential internal commands, even if they are "protected" by the presence of the `@` character; when classes and packages are written that protection is not effective. Nevertheless it might have been a good idea to illustrate the right technique for checking the existence of the command to be (re)defined so as to avoid messing up the whole system; the `ifthen` package allows to check if a control sequence is undefined, but at the class or package level even the kernel macro `\@ifundefined` can be used.

There are other primitive TeX structures that can be very useful in producing reliable code also for LaTeX, but I won't insist on this point since it would carry me too far.

Let me reach the conclusion. The second edition of *The LaTeX Companion* is a must for every serious LaTeX user. It complements Lamport's handbook, but it adds such a great amount of useful information and wealth of working examples that any user will find it invaluable. Moreover, these examples carry with them the authors' great experience, and let's recall that most of them are members of the LaTeX 3 team — who could know LaTeX better than they? The few errors or omissions are not really so essential, provided the user keeps a little warning light blinking in the background of his/her mind.

When ten more years have elapsed, perhaps we'll see a third edition that will be even more useful, for the future LaTeX of the year 2014. I hope so, because ten years from now I'll still be using LaTeX, for which I thank not only Leslie Lamport, who started the whole game, and the contributors of the many extension packages that enhance so much the basic typesetting interpreter and engine created by Donald Knuth, but also all the authors of this excellent book.

References

[1] Mittelbach, Frank, Errata list for *The LaTeX Companion, Second Edition*, `http://www.latex-project.org/guides/tlc2.err`.

[2] Syropoulos A., *Digital typography using LaTeX*, Springer Verlag, New York, 2002.

◇ Claudio Beccari
Politecnico di Torino
Turin, Italy
`beccari@polito.it`

Book review: *Guide to LaTeX, 4th Edition*

Douglas Waud

Helmut Kopka and Patrick W. Daly, *Guide to LaTeX, 4th Edition*. Addison-Wesley 2004, pp. 624, ISBN 0-321-17385-6, USD 49.99.

I believe I should point out at the beginning that my wife has a t-shirt which reads "My next husband will be normal". However the problem is deeper than that. I seem to be attracted to groups where the whole membership is not normal, for example, the local Linux users group. Reading the Kopka and Daly book I have come to the conclusion that one should not consider TeX users to be normal either. And I think it is important to keep this in mind.

This whole issue arose when I started by looking at the cover and introductory material to see what the authors' objective was. I found "how to begin using LaTeX" right on the back cover. Against this backdrop I began to read. By chapter 2 I began to get the sense of being completely overwhelmed if I tried to put myself in the mind set of a neophyte. I can't imagine a normal person not completely boggling soon after diving into this book. As a specific example, when I reached chapter 1, section 1, subsection 1 (i.e., the very beginning) I encountered the word "markup" in the first line of the first bullet. As a test, I gave that paragraph to my wife to read and she confirmed my suspicion that normal people do not know what that word means.[1] I then took her forward two pages to where the authors give an clear example of markup in both HTML and plain TeX and she now saw right away what the term meant. But I would still argue that something was basically wrong. That first bullet could have been worded without the jargon. More importantly, I suspect such stumbling blocks for the beginner would not have survived if the authors had bounced the book off test readers who were *completely* naive when it come to TeX/LaTeX.

I was reinforced in this view that TeXers are not normal when I joined the TeXhax list. While there will always be those who simply do not RTFM there is still a large group who seem simply bewildered by the jargon (RTFM, for example). While this soon wears off, it is still very real initially. The strategy of teaching people to swim by pushing them off the dock does not work well here. It is better to recognize that total immersion can create an aversion — the reverse of what is intended. I find a parallel situation with those new to Linux. Again there seems to be a steep learning curve. Here the neophytes forget that they have gone through a similar process coming to grips with Windows or Microsoft Office. However, reading a periodical like *PC Magazine*, where page after page is devoted to clarifying yet another dark corner of the Microsoft world, demonstrates that neither Linux or LaTeX is really that difficult once you put it into that perspective. However, one must not ignore this barrier; although it may be misperceived as being high, that perception is still very real and must be kept in mind if one is going to proselytize successfully.

Now, as will become clear below, I still found this book a very useful reference. So, on the assumption that a "newbie" or two may still try to use it as a primer, I shall give some advice[2] and list a few general tricks that the neophyte should know. My authority for doing this is that, over the years, I have made just about every mistake imaginable, so I am quite familiar with the pitfalls, if not the solutions.

- Crawl before you walk. Do not try to create a fancy/complex document first time around. Start with a "Hello World" example as the first step (this is a standard computer ritual to show you have something working; you write just sufficient code to have the program capable of saying "Hello World" back to you). In fact, Kopka and Daly show you how to do just this on page 13. Now, emboldened by this success, add *o*ne new frill; for example, put your name on a line below the "Hello" line but at the right hand end thereof. Next you might try to put "Dear sir" at the left end of a line above the "Hello". As you can see, we are on our way to creating a letter. When you finish your creation, go to Kopka and Daly and see how they do it; at this point you will be in a much better position to appreciate more of the fine points.
- At each stage in the preceding process, run the file through LaTeX, and look at the output file with a viewer. Since you are making just one change at each stage, it is reasonably easy to locate any typos and the like. Also, in no time, this repetition will become second nature and you'll be hooked.
- Don't be afraid to make mistakes. That is really the way you learn, not by reading and somehow

[1] In fairness, I should add that she, an anesthesiologist (our kids used to say "Mommy is the *real* doctor"), was wondering why I had suddenly taken an interest in the problems associated with rubber gloves — allergies and the like — so much so as to buy a book on latex!

[2] At my age you tend to do this a lot.

(osmosis?) having it all sink in without any effort on your part.

- Use Kopka and Daly, or a clone, as a reference book. Don't try to read it like a novel. One can skim to get the lay-of-the-land but, in the end, the stuff sinks in only when you actually use it.
- Look in books for specific examples of code (you will find Kopka and Daly are great here). This way, you are less likely to leave out a key space or the like when you are still new to the game and do not yet appreciate all the nuances. With time, you will begin to see patterns and what initially may seem arbitrary will suddenly make perfectly good sense. It is rather like learning to play bridge. Initially you are amazed at how people can figure out where cards must be and then one day you suddenly realize "the Queen has to be in that hand" and, indeed, that it where it turns out to be!
- Sometimes you find that simple trial-and-error is quicker than pausing to look something up. Current computers are fast enough that, especially for short documents, one can rapidly type, LaTeX, and view to get fast feedback. And it is hard to go the trial-and-error route without learning something. (Some may disagree with me here but you should realize their problem — they're normal.)
- Don't be overwhelmed by the huge number of commands and variants thereof that are available. As an academic pharmacologist, I have had to face this same problem with medical students trying to come to grips with "all those drugs". I point out that this is an advantage, not a disadvantage; the more you know, the finer the control you can exert. Their patients will appreciate being able to tailor drug choice individually. Similarly, the breadth of LaTeX means you can do all sorts of wonderful things. With LaTeX however, you are better off than the medical students. They don't know what the next patient will need. You, on the other hand, can choose when to go down a new path.
- At each stage do not be afraid to look a little farther than you feel is friendly territory. Initially such forays will be into *terra incognito* but each time you will find the critters you encounter are a tad less frightening and eventually you will find yourself looking at the code of a package file and actually beginning to be able to make sense of it. The trick is not to try to do it in one swell foop.
- In the words of Boston's Tom Lehrer [6], plagiarize. With computer programming, you never want to reinvent the wheel. What you want to do is find someone's code that comes close to what you want and then make a relatively minor modification (in this case, you can crawl without ever having to learn to walk). To this end, a book like Kopka and Daly comes into its own. Now all that detail is no longer just detail; it is code you can build on. This is why it is so useful to have a book with lots of examples. You can also see why there is such enthusiasm for "open source" code (software like Linux and LaTeX which give you not only the program, but also the underlying instructions which make it work; thus everyone can "look under the hood" both for clarification and as a base for further development). You become part of a large community; sooner than you may realize, you will create something someone else will find useful and you will have become part of the system.
- Take a look at Peter Flynn's [1] recent miniguide to LaTeX. It is directed specifically at the newbie.
- Try to set aside a little time to play, to dig a little deeper now and then — all work and no play ... and all that. I get no kicks from computer games but I can derive a lot of entertainment out of deciphering how a snippet of code works and/or how to get it to work for me. Try it; you may get in touch with your abnormal self.
- How do you pick which books you will keep nearby? First you *do* need something at hand. One way to customize the selection is to wait until you have a specific problem, preferably similar to most that you stumble over, and go to the book store to see how the various candidates fare when faced with a specific problem.

(I cannot get anything right; here, instead of a book review, I go off on a tangent writing a guide to getting started with TeX.)

I am now going to proceed by abandoning that initial goal of using the book as a "Gentle Introduction" (to coin a phrase) and refer a reader who is really new to LaTeX to a less detailed first round such as Flynn's opus mentioned above, or the original "bible" by Lamport [5].

So I shall start all over again. I found Kopka and Daly's book very useful and I suspect that the rest of you readers, who I still suspect are not normal, will also find it valuable.

This book is in fact part of a series of four books and is written to take advantage of this linkage. Thus, the authors can refer to other members of the series when it is impractical to cover all the details

without getting off track. For example, they can refer to the *LaTeX Graphics Companion* [3] to take over where the present text leaves off when dealing with graphics. Similarly, the HTML aspect can be found in expanded form in the *LaTeX Web Companion* [2]. And all three are scions of the original LaTeX book by Lamport[5] and of another general book in the series [7]. They all fit together in a reasonably integrated whole. This collection, in turn, stands on the foundation of TeX as originally set forth by its god [4].

So what specifically does Kopka and Daly offer? First, lots of examples. As noted above, this makes life much easier, even for experts. Especially when you are trying to get a specific job done, you want something you can plug right in and get on with the main task.

A simple perusal of the table of contents gives a rough view of the lay of the land. In particular, of the *current* view. LaTeX is continually being improved. While this can be a nuisance — you just get the hang of it and they change it — it is good in the long run. The tool becomes more powerful. Again the price you pay for this power is having to come to grips periodically with new toys.

As a notable example, the last few years have seen the rise of PDF files[3] as the *de facto* standard for final presentation of documents, especially in an electronic form. Thus the (La)TeX gurus have given us pdfLaTeX, a version of LaTeX which can generate PDF files directly. This opens the door to use of hyperlinks, and we are served the `hyperref` package to take care of the details. The reader who may have noticed allusions to such developments but has not yet had time to look more closely will find Kopka and Daly very useful. This edition arrives just at the right time (the PDF world seems to have stabilized enough to be ready for prime time and the rest of us) to bring us up to date in this area.

The beginner will like the section *TeX and its offspring* as a tidy map of who's who. If you are starting cold, little things like why there is a LaTeX, a LaTeX 2.09, and a LaTeX 2_ε, can be confusing. (The situation is similar in Boston where visitors find the main streets are not labeled "Because everybody will know where they are!").

[3] A file with the extension `.pdf` signals "PDF format", that is, it follows the rules for Adobe's "Portable Document Format" which has been designed to make portable files with the ability to present pictures, sounds, hyper-references (links to targets outside the current document) and the like. The Adobe people did such a good job (as they did with the predecessor "Postscript") that PDF has been widely accepted. One program for reading such files, Adobe Acrobat, has been made available by Adobe at no charge.

I have not methodically listed chapter topics. They are about what you would expect in a systematic text. I found the details consistently clear for anyone who has some background in (La)TeX and who will therefore not stumble on terms like "source" or "bounding box". I noticed no obvious big gap but, and I am probably dating myself here, I did miss PiCTeX (although it did get into the *Graphics Companion*, if not its index.) I did not find a note on how to pronounce LaTeX; I don't know whether this is good or bad.

I would say the book's main strength is the gathering together in one place of an up-to-date summary of what one can do with LaTeX and its clones. In this spirit, about one third of the book is devoted to appendices. For example, Appendix G contains an alphabetical list of commands with both brief descriptions of their behavior as well as references to both section and page where the fuller treatment appears. A second section of G gives tables showing a wide range of symbols you might want and how to generate them. The first appendix puts the New Font Selection Scheme into perspective — another topic where a reader who has not been keeping up-to-date will welcome a primer. Appendix C gives a systematic approach to error messages including lists of the various possibilities. Appendix D delves into "LaTeX Programming". Most users probably will not think of themselves going down this path but they may still find this survey useful if they want to make a minor tweak to a package. Take a peek.

As you may gather, I think readers will find this book useful. Are there any flaws? A glossary would have been handy, for example when Chapter 2 starts with a reference to a "source file". (Yes, if you know what that means, you have passed a simple test to demonstrate that you, too, are not normal!) When the reader first hits "bounding box" not only is a glossary missing, the term did not make the index. I can nit-pick and wonder why a section starts with the title "Chemical formulas and boldface in math formulas" and then discusses them in the reverse order. Overall, such issues are well offset by conveniences like a list of "Mathematical typesetting conventions" which, while not LaTeX *per se*, puts the latter into good perspective.

The book includes a compact disc with selections from the TeX Live collection — enough to enable installation of a working system on Windows, Linux, or a Mac OS X system (Linux users will generally find their installation already provides (La)TeX albeit of uncertain vintage).

References

[1] Peter Flynn. Formatting information. *TUGboat*, 23(2):115–237, 2002.

[2] Michel Goossens, Sebastian Rahtz, Eitan Gurari, Ross Moore, and Robert Sutor. *The LaTeX Web Companion*. Addison-Wesley, Reading, MA, 1999.

[3] Michel Goossens, Sebastian Rahtz, and Frank Mittelbach. *The LaTeX Graphics Companion*. Addison-Wesley, Reading, MA, 1997.

[4] Donald E. Knuth. *The TeXbook, Computers and Typesetting, Vol. A*. Addison-Wesley, Reading, MA, 1986.

[5] Leslie Lamport. *LaTeX: A Document Preparation System*. Addison-Wesley, Reading, MA, 2nd edition, 1994.

[6] Tom Lehrer. Lobachevsky. In *Songs by Tom Lehrer,* track 6. Tom Lehrer, Cambridge, MA, 1952. 33 rpm 12 inch record, US$3.95, "plus 30 cents shipping".

[7] Frank Mittelbach, Michel Goossens, Johannes Braams, David Carlisle, and Chris Rowley. *The LaTeX Companion*. Addison-Wesley, Reading, MA, 2nd edition, 2004.

◇ Douglas Waud
Department of Pharmacology
University of Massachusetts
Medical School (retired)
17 Lantern Lane, Shrewsbury, MA, USA.
douglas.waud@umassmed.edu
http://users.umassmed.edu/douglas.waud/

Book review: *Guide to LaTeX, 4th Edition*

Mimi Burbank

In an attempt to represent the "new user" end of the spectrum — I've used *Guide to LaTeX* since it first appeared on the market. However, I begin at the back of the book — having done so since I was first introduced to TeX way back in 1985. I could spell the word "computer" at that time, and was hired simply because I was too naive to understand just how complex TeX was going to be. I had never even seen a computer before. I think I spent three months reading the index (slept with it under my pillow at night) and then going back to the front of the book and reading chapter by chapter.

For someone who has been introduced to LaTeX but has not had the time to spend learning its intricacies, this is an excellent reference book in that you can look in the index for the word (since most commands are a backslash and some semblance of the actual "word" itself) and then choose whether you wish to do the "quick and dirty" (just need to know how many options and which ones they are), or whether you need a deeper understanding of the definition. Finally, while the book is not intended to represent the "full" instructions for "every" package, the authors have provided pointers to the best information available for various packages and utilities on the web and on paper.

For typists who have to either type or edit manuscripts, it presents a comprehensive reference utility. It is updated often enough to be "current" and is written in lay language.

I confess I consider it required for all of the people that I work with, who need to know how to do anything with LaTeX.

◇ Mimi Burbank
CSIT/FSU
Tallahassee, FL 32306-4120
mimi@csit.fsu.edu

Abstracts

Editor's note: This issue of *TUGboat* contains translated abstracts and summaries from recent publications by several other TEX user groups. For a complete list of all user group publications, see `http://tug.org/pubs.html`.

Les Cahiers GUTenberg is the publication of the French language TEX user group, GUTenberg. Their web site is `http://www.gutenberg.eu.org`.

Les Cahiers GUTenberg
Contents of Issue 42
(July 2003)

Jacques André, Éditorial [Editorial]; p. 3

The editor opens with a description of factors which have contributed to the long gap between this issue and the previous one, no. 41, which appeared in November of 2001: other large issues not yet finished, conference proceedings which have taken a long time to produce, the very utility of even having a hardcopy publication in the face of the Internet, a lack of time, a lack of articles ... Nevertheless, there will be a return to a regular schedule.

To get back on track, this is a small issue of 64 pages, which actually used to be the norm. There are two TEX articles (a technical one by Jonathan Fine and an historical piece by Jean-Michel Hufflen), and one rather more typographic in nature (by Markus Kohm). The Fine and Kohm articles had appeared elsewhere, in English and German, respectively.

The editor concludes by thanking the various members of the *Cahiers GUTenberg* production team (Daniel Flipo, Bernard Gaulle and Gilles Perez-Lambert), as well as the translators (Jean-Marie Hufflen of Fine, and Julie Le Boulanger of Kohm).

Markus Kohm, Étude comparative de différents modèles d'empagement [Comparative study of different page make-ups]; pp. 4–25

How to organize text onto a page is an art refined by professionals since the middle ages. Through time, numerous techniques have been constructed, challenged, forgotten, but also developed and enriched. Some of these methods contain expressions which have become legendary. And even though they say that myths include some kernel of truth, their origin can also present some perils. It's therefore important to know how to distinguish between myth and truth.[1]

[Translation of author's abstract]

Jonathan Fine, Appeler TEX comme une fonction [TEX as a Callable Function]; pp. 26–37

Traditionally, TEX is run as a batch program. However, TEX can also be run as a daemon, with a callable function interface. This talk describes the opportunities and problems that follow from this new use of TEX.[2]

[Author's abstract]

Jean-Michel Hufflen, Mes diverses périodes avec LATEX [My various bouts with LATEX]; pp. 38–60

This article outlines the different stages of an experience with LATEX. Some details are largely autobiographical; however, evoking successive work places allows the author to show how LATEX was used and what problems existed.

[Author's abstract (edited)]

Bernard Gaulle, Rectificatif : précisions sur la francisation [Correction: Clarification on 'francisation']; p. 61

In February of 2003, a French translation of *A Short Introduction to LATEX 2ε* was produced, to commemorate 25 years of TEX. This brief one-page item clarifies the development and naming of Gaulle's various style files for French texts.

[Summary of note]

– – * – –

Articles from *Cahiers* issues can be found in PDF format at the following site:

`http://www.gutenberg.eu.org/publications`

[Compiled by Christina Thiele]

[1] This text originally appeared in German, in *Die TEXnische Komödie*, Dec. 2002, no. 4 (pp. 28–48). The French translation was done by Julie Le Boulanger.

[2] This text originally appeared in English, in the 2002 EuroTEX proceedings (pp. 26–30). The French translation was done by Jean-Michel Hufflen.

Editor's note: *MAPS* is the publication of NTG, the Dutch language TEX user group. Their web site is `http://www.ntg.nl`.

***MAPS* 27, Spring 2002**

data, as markers, lines, and areas. While fairly simple, they have been successfully harnessed to appear in a wide range of real-life applications, up to logarithmic graphs and (with some patience) complex multipanel displays. Of course, the macros are a tool for drawing final graphs rather than exploring or transforming data sets. [Author's abstract]

Hans Hagen and Ton Otten, Figures; pp. 36–40

Within the TEX community there is a widely used database for bibliographic references, BibTEX, but not for figures. To manage figures ConTEXt now supports a figures database. The database is set up in XML and converted to an interactive PDF figure library featuring ordered displays and a search mechanism. From the library, figures can be included easily in ConTEXt documents as long as both the PDF and the XML files remain present.
[Authors' abstract]

Ernst van der Storm, DTP'en met LATEX [Desktop publishing with LATEX]; pp. 41–44

Report on the use of LATEX for desktop publishers showing some simple macros and how to incorporate images: a practical description.
[Translation of author's abstract]

Karel H. Wesseling, Gertrude L. van der Sar and Jos J. Settels, From PC-Write to ConTEXt; pp. 45–50

A tale of more than 10 years of joy and struggle with TEX followed by a period of bliss, of easy to use tools, quickly obtained results, and incredible possibilities from the coming of 4TEX and ConTEXt, narrated by non-gurus. [Authors' abstract]

Karel H. Wesseling, A do-it-yourself `thebibliography` in ConTEXt; pp. 51–55

Moving from LATEX to ConTEXt is not really simple, but to return from ConTEXt to LATEX would have been equally hard were it not for a publication by Berend de Boer in MAPS 24 explaining how to do LATEX things in ConTEXt. Only one thing was missing, a do-it-yourself `thebibliography`. Hans Hagen had a solution which is described below.
[Author's abstract]

Siep Kroonenberg, TEX voor thuis [TEX at home]; pp. 56–59

A beginner's column, which in this issue discusses installing a TEX distribution on various popular platforms.

Siep Kroonenberg, Mac OS X als TEX platform [Mac OS X as a TEX platform]; pp. 60–61

Now that the Macintosh platform has been converted to UNIX and has built-in support for PDF, it has good credentials as a platform for TEX. The TEXShop program is proof of this.
[Translation of author's abstract]

Siep Kroonenberg, Juggling `texmf` trees; pp. 62–65

`texmf` trees can make a TEX installation more maintainable. With creative use of environment variables, it is possible to run different versions and different configurations in different xterm or console windows. [Author's abstract]

Hans Hagen, MathML; pp. 66–119

It is a well known fact that TEX can do a pretty good job on typesetting math. This is one reason why many scientific articles, papers and books are typeset using TEX. However, in these days of triumphing angle brackets, coding in TEX looks more and more out of place.

From the point of view of an author, coding in TEX is quite natural, given that some time is spent on reading the manuals. There are however circumstances where one wants to share formulas (or formula-like specifications) between several applications, one of which is a typesetting engine. In that case, a bit more work now saves you some headaches later due to keeping the different source documents in sync.

In the following we will discuss the mathematical language MathML with respect to typography. As a typesetting vehicle, we have used ConTEXt. However, the principles introduced here and the examples that we provide are independent of ConTEXt. For a more formal exploration we recommend the MathML specification.
[Author's introduction (edited)]

MAPS 28, Fall 2002

Johannes Braams, Redactioneel [From the editor]; p. 1

Overview of the issue's contents, including an apology for unexpected results in the last issue. Corrections are included in this issue.

[Compiled by Steve Peter]

Editor's note: *TEXemplares* is the publication of CervanTEX, the Spanish TEX user group. Their web site is `http://www.cervantex.org`.

TEXemplares #4, 2003

José Martínez de Sousa (typographer, orthographer, lexicographer, and bibliologue), Algunos problemas de ortotipografía [Some problems of orthotypography]; pp. 7–14

For typographers, orthography presents two fairly differentiated sides: on the one hand, what we call *usual orthography*, which we all claim to know for the unfolding of our daily life regarding written comunication, and *technical orthography* (graphic rules for scientific and technical elements), that comprises the *scientific orthography* (rules of scientific writing) and *typographic orthograpy*, or *orthotypography* (rules for the writing of graphic elements). Here we are especially concerned with orthotypography, whose current situation reaches worrisome levels, thanks undoubtedly to the application by Tirians and Trojans of rules or pseudo-rules that are neither generally known nor approved by experts, who on the other hand fail to apply those rules that we have known and acknowledged for centuries. To this, as could be expected, has been added the introduction of self-editing, especially since 1985, due to the facilities that it offers to the user of a computer, and of software that gives him in his inexpert, but not less enthusiastic and indefatigable hands, all the possibilities of graphic expression. . . , but, unfortunately, not the knowledge necessary for the application of those potentials with maximum accuracy.

If we analyze the *Academia*'s[1] intrusions into the field of orthotypography, especially as seen in its *Ortografía de la lengua española* of 1999, we see that they bear no little incoherence and disagreement with the uses and customs of a world that the *Academia* has never wanted to pervade and into which now, because of lack of specific knowledge, is introducing confusion. Let us see some cases.

[The "cases" are the use of quotation marks, the period in conjunction with closing signs, orthotypographical Anglicisms imported by the *Academia*, italics in conjunction with roman, and the placement of footnotes.]

[Translation of author's introduction]

[1] Translator's note: The *Real Academia de la Lengua Española* is the central regulator of the norms and uses of the Spanish language.

Enrique Meléndez Asensio, Uso de fuentes TrueType de Microsoft con TEX [Using Microsoft TrueType fonts with TEX]; pp. 15–23

This contribution describes the use of TrueType fonts with TEX and LATEX. It is based on the document "Using TrueType fonts with teTEX and dvips", written by Harald Harders, available at

`ftp://ftp.dante.de/tex-archive/info/TrueType/ttf-tetex.pdf`

[Translation of author's introduction]

Roberto Herrero, Una breve reseña de MetaPost [A brief review of MetaPost]; pp. 30–33

Figures created with any graphic application can be easily included into LATEX documents using any tool that converts the result to the PostScript format and with the `graphicx` package.

Among those applications there are some that, because of their characteristics, are especially suited for use with LATEX. Among them it is good to mention:

- The `picture` environment provided by LATEX macros themselves.
- The program `xfig`, which allows (LA)TEX text within the figures, which is especially useful to introduce mathematical formulae.
- The program `gnuplot`.

To the mentioned tools should be added the program that concerns us now, MetaPost, which, because of its peculiar origin and capabilities, is possibly the most highly recommended tool to incorporate mathematical graphics into LATEX.

[Translation of author's introduction]

[The article has the following sections: 2. Origin of MetaPost; 3. Basic characteristics of MetaPost; 4. MetaPost packages; 5. Metagraf.]

TEXemplares #5, 2003

José Luis Díaz de Arriba, Presentaciones LATEX: un enfoque simple usando FoilTEX [Presentations in LATEX: a simple approach through FoilTEX]; pp. 4–22

The use of devices to project the video output of a computer onto a screen become more and more common every day, making obsolete the old method of vinyl slides and projector. In LATEX it is possible to generate PDF, which can be viewed in full-screen mode with Acrobat Reader and other PDF readers, and this gives us the chance to do presentations without leaving our favorite software. Actually, no special package is required to do a presentation with LATEX. The only requisite is to set the paper size

to the screen's proportions, and to use a large and readable type font. However, if "special effects" are desired, such as background graphics, slide-to-slide transitions, gradual definition of text or graphics, etc., using a package can be helpful. In this article I share my personal experience, and some of the solutions I have come up with.

[Translation of author's abstract]

ÁTOPOS, Reseña de LaTeX para las Humanidades [Review: *LaTeX for the Humanities*]; pp. 23–36

This is an out-of-the-ordinary review. To begin with, it is written by the author himself (the enigmatic *átopos*). And it is fiction. It is in the form of a dialog — characters are Don Quixote, Tux, Socrates, and K-Nut. They talk about a mysterious document, *LaTeX for the Humanities* ("LaTeX para las Humanidades"), which by the way is available at:

`http://rt0016xp.eresmas.net/lplh/latex-humanidades.pdf`.

The book itself is, as the author describes it, a "historical-mythological-computer *divertimento*", whose protagonist is Tux, and is intended to give a "practical, elemental, and entertaining introduction to LaTeX to authors in any field of the humanities". The review goes beyond the description of the book, and addresses "not only the *what*, but the *why* of the book". Sections include "Socrates and wisdom", "K-Nut and beauty", "Tux and freedom". Both review and book are worth reading.

[Summary of note]

***TEXemplares* #6, 2004**

JOSÉ M. MIRA, Bibliografía flexible: el sistema `flexbib` [Flexible bibliography: the `flexbib` system]; pp. 8–26

Automated processing of bibliographies with BibTeX offers an important level of comfort for the user, provided that the bibliographic model used is one of the standard styles, and, furthermore, that the user writes in English. But the adjective 'standard' is actually a euphemism, because the list of styles to be found at CTAN is endless... and it is fairly easy to be lost in that forest before finding the sought-for solution. Surprisingly, and at odds with the level of standardization and flexibility developed in other aspects of LaTeX, bibliography processing has not reached the status of being accessible to new users, and basic issues, such as language handling, have yet to be automated.

This text advances a proposal to improve this situation, and contributes some tools to carry it out. A system is used that allows for standardized and flexible processing, including language and a wide variety of parameters that simplify bibliography customization.

[Translation of author's abstract]

FRANCISCO J. ALCARAZ ARIZA, LaTeX, Linux y la Botánica: una excelente combinación [LaTeX, Linux, and botany: an excellent combination]; pp. 27–40

...A review of Spanish-language writings on botany reveals the fundamental reality that the use of operating systems different from those that have their headquarters in the rainy city of Seattle [i.e., Microsoft Windows], and the use of text processors other than MS Word, not to mention non-WYSIWYG applications, is an act of fantasy...

This is certainly a discouraging state of affairs, but we hope it will start changing in the near future; we believe there are signs in the environment that will give free software a greater role in the world of botany...

The sections of this article present our experience with using Linux and LaTeX for teaching and scientific research in botany, as a proof that it is possible to use alternative ways, and that the latter, against-the-tide at first, render much better performance...

[Translation of author's introduction (edited)]

SALVADOR SÁNCHEZ-PEDREÑO GUILLÉN, Edición de partituras [Score editing]; pp. 41–71

This text constitutes a brief introduction to musical score editing in TeX and its environment. It focuses on MusiXTeX, although brief references are made to packages, pre-compilers, and graphic environments more or less connected to MusiXTeX.

[Translation of author's abstract]

[Compiled by Federico Garcia]

Reports

Report on the Pune workshop on LaTeX and free mathematical software (July 2003)

S. A. Katre, Manjusha Joshi (coordinators)

A workshop on LaTeX and free mathematical software was organised by Bhaskaracharya Pratishthana (BP), Pune, India, during 9–14 July, 2003. It was mainly sponsored by the international TeX Users Group (TUG) with a generous grant of USD 2100 (Rs. 99,036), as a joint activity with the Department of Mathematics, University of Pune, and was partially supported by the DSA programme of UGC in the department.

The focus of the workshop was on the LaTeX software for mathematical typesetting. In addition, a number of other free mathematical software packages were introduced to the participants. The workshop was organised for participants from the educational field (mainly from universities and colleges) with some knowledge of TeX and a background in mathematics.

This is the first workshop of this kind organised in India which dealt with TeX topics as well as free mathematical software running on Linux. The participants were very enthusiastic and many of the participants worked on into late nights to practise and assimilate various topics on LaTeX and mathematical software.

Organising committee

Chairman: C. S. Inamdar, BP.
Co-ordinator: S. A. Katre, Dept. of Mathematics, University of Pune.
Co-ordinator: Manjusha Joshi, BP.
Member: Amitabh Trehan, M.G.A.H.V., Delhi.

Speakers and topics

1. Mr. Amitabh Trehan (IIT, Delhi): Linux, Pdfscreen, Devnag, LaTeX: indexing, table of contents.
2. Dr. S. C. Phatak (IoPB, Bhubaneswar): Gnuplot, Xfig.
3. Dr. Surendran (Pune): TeXmacs, WIMS.
4. Dr. S. A. Katre (Univ. of Pune): TeX utilities.
5. Ms. Manjusha S. Joshi (BP, Pune): PStricks, POV-Ray, references in LaTeX.
6. Dr. Ajit Kumar (St. Xavier's College, Mumbai): PSfrag, DCPic, MuPAD.
7. Dr. A. V. Jayathan (TIFR, Mumbai): Macaulay 2.
8. Dr. M. S. Bakre (Univ. of Mumbai, Mumbai): Scilab, Euler, Octave, software installation.
9. Mr. Sumit Bharadwaj (IIT, Delhi): MathML, Prosper, BibTeX.
10. Mr. Niyam Bhushan (Delhi): Linux, publishing workflows, typography, image enhancement & conversion, Plone.

In addition, on 8th July, before the workshop proper, for some participants having a limited background in TeX, a lecture by Amitabh Trehan to introduce the basic concepts in TeX was arranged. There were also practice sessions based on exercises in LaTeX prepared by S. A. Katre and Manjusha Joshi.

Software provided

Registered participants were given two CDs, notes on various topics discussed in the workshop, a file with register of 100 pages, a pen, a few blank papers, badge and *LaTeX Tutorials: A primer to LaTeX 2ε*, prepared by the Indian TeX Users Group (Editor: E. Krishnan), 2002.

The CDs contained the following software:

1. TeX Live version 7 (containing Devnag, PStricks, Pdfscreen, PSfrag, PSplot, etc.).
2. POV-Ray, MuPAD (light version), Gnuplot (for Windows), Macaulay 2, CoCoA, Maxima, TeXmacs, WIM Server, Scilab, Euler, LaTeX suite, MathML, TeXnicCenter.

Some software discussed in the workshop was already available in Red Hat Linux, such as Xfig, GCC, Gnuplot, Octave.

The notes given to participants included installation of the software on the CDs, assignments on TeX, introductory material on LaTeX, etc.

Most of the mathematical software on the CD was discussed during the workshop. While MuPAD is general purpose mathematical software, Macaulay 2 and CoCoA are specialized software for algebra. Gnuplot and Xfig are useful for drawing figures. The insertion of the figures drawn using this software in LaTeX documents was discussed during the workshop. TeXmacs is a front end which gives LaTeX output for some of the mathematical software, such as Maxima, POV-Ray, Gnuplot, SciLab, etc. Browsing by subject and education level is possible through the WIM Server for MuPAD, Maxima, Macaulay 2, PARI/GP, GAP, Yacas, Octave, etc., in which output of the software is TeX-formatted text. A mathematical document prepared using LaTeX can be put on the net using utilities such as Itex2mml, by conversion to MathML.

Information on participants

Participants were from 18 institutes and universities located throughout India. There were in all 41 participants, out of whom 26 were outstation and 15 were local. There were in all 9 female participants. The participants outside Maharashtra State came from Goa, Delhi, Chennai, Chandigarh, Meerat, Ahmedabad, Daman, Dharawad, Patan, Jhansi, Darjeeling, Agra, Patiala. Participants within Maharashtra came from Pune, Mumbai, Aurangabad, Sangamner, Verdha, Shegaon. Outstation participants and Speakers were accommodated at SET Guest House, Univ. of Pune and BP Guest House. The local participants comprised of past students (now teachers in various colleges in Pune) and current students of the Department of Mathematics at the University of Pune. The full roster of participants is included at the end of this report.

The participants were given a certificate of participation with a logo of the TEX Lion.

Technical arrangements

A total of 20 computers were arranged for practicals, with an LCD projector for lectures and demonstrations etc. One classroom was temporarily converted to an additional computer lab, in which 14 computers were arranged. Six computers were arranged for practical work in the main computer lab. Other than these, three computers were available for speakers for their work regarding the workshop, Internet and mail. Nine computers were on a LAN and were also connected to a laser printer. Printing facilities were provided to the speakers.

Parallel activities

PLUG Linux Stall: In the city of Pune, a local group called the Pune Linux User Group helped us with installation of Linux and other software on the temporary machines. They also put their stall at the institute, where they made Linux CDs available at a nominal charge. About 20 participants took advantage of this arrangement.

Springer Book Exhibition: Springer, one of the leading publishers of scientific books, took advantage of the opportunity of the workshop. They exhibited books on LATEX, typography, MuPAD and several other topics.

Acknowledgements

The organising committee is grateful to the international TEX Users Group for the timely and generous grant of USD 2100 without which it would not have been possible to hold the workshop at the national level.

Some information regarding possible topics and books for the workshop, especially related to LATEX, came from the mailing lists `tugindia@tug.org` and `indic-dev@tug.org`.

We are thankful to the eminent scholars Mr. C. V. Radhakrishnan (Trivandrum), former secretary, TUGIndia, and Dr. E. Krishnan (Trivandrum), Chairman, Free Software Foundation, India, who encouraged the organisation of the workshop and helped in the initial stages. Dr. Kaveh Bazargan (Trivandrum & U.K.), member, TUGIndia board, also helped in various aspects of the workshop.

We are thankful to Prof. A. S. Kolaskar, Vice-Chancellor, Univ. of Pune, and Prof. N. S. Bhave, Head, Dept. of Mathematics, Univ. of Pune, to consider the workshop as a joint activity with the department.

The support from PLUG, Pune, regarding Linux installation requires a special mention. We thank Mr. Bhamburkar who made available seven diskless machines for the workshop in the nick of the time, without any fee. We also thank Mr. Sudhanwa Joglekar, Dr. Surendran, and Dr. Gangawane who helped in the computer related matters, and Prof. Kalidoss for providing copies of the book *LATEX Primer*, prepared by TUGIndia.

We thank the office staff of BP: Ms. Veena Kulkarni, Mr. S. R. Gosavi, Mr. Sunil Sawant and also the Trustees of Bhaskaracharya Pratishthana for all the help they extended for the organisation of the workshop.

◇ S. A. Katre, Manjusha Joshi
(coordinators)
Bhaskaracharya Pratishthana,
Pune, India
`bhaskara_p@vsnl.com`
`http://www.bprim.org`
`http://education.vsnl.com/bp/texwork.html`

Appendix A Participants

Outstation participants:

1. Sartaj Ul Hasan, I.I.T. Bombay, Powai, Mumbai.
2. Mr. Malay Kumar Ghosh, North Bengal University, Darjeeling.
3. Mr. Partha Sarathi Debnath, North Bengal University, Darjeeling.
4. Dr. Sanjeev Kumar, Institute of Basic Science, Agra.
5. Mr. Bhatoa Jogindar Singh, Govt. College, Daman.

6. Mr. C. S. Salimath, Karnatak University, Dharwad.
7. Sanjay Choudhary, Institute of Basic Science, Khandari-Agra.
8. Dr. Manjusha Gandhi, Shri Sant Gajanan Maharaj College of Engineering, Shegaon.
9. Mr. Narahari A. Patil, Shri Sant Gajanan Maharaj College of Engineering, Shegaon.
10. Mr. Rupen Pratap Singh, Ch. Charan Singh University, Meerut.
11. Dr. A. K. Desai, Gujarat University, Ahmedabad.
12. Mr. Udayan Prajapati, St. Xaviers College, Ahmedabad.
13. Mr. Habeeb Basha Syed, I.I.T. Bombay, Powai, Mumbai.
14. Mr. Arunkumar Patil, I.I.T. Bombay Powai, Mumbai.
15. Mr. Subhendu Bhanti, North Gujarat University, Patan.
16. Ms. Anju Rani Gupta, Bundelkhand University, Jhansi.
17. Dr. Avanish Kumar, Bundelkhand University, Jhansi.
18. Mr. Israr Ahmad, Jamiya Millia Islamiya, New Delhi.
19. Dr. Y. S. Valaulikar, Goa University, Goa.
20. Dr. Sushil Kumar Tomar, Panjab University, Chandigarh.
21. Mr. Rajesh Agarkar, Mahatma Gandhi Antarrashtriya Hindi Vishwavidyalaya, Wardha.
22. Ms. Hema Godbole, Mahatma Gandhi Antarrashtriya Hindi Vishwavidyalaya, Wardha.
23. Dr. H. S. Kasana, Thapar Institute of Engineering and Technology, Patiala.
24. Dr. S. K. Panchal, S.N. Arts D.J.M. Commerce and B.N.S. Science College, Sangamner, Dist. A. Nagar 422 605.
25. Shibsankar Karmakar, North Bengal University, Darjeeling, 734430.
26. Malay Kumar Ghosh, North Bengal University, Darjeeling, 734430.
27. Dr. S. K. Nimbhorkar, Dr. B.A. Marathwada University, Aurangabad-431 004.

Local participants:

1. Mr. Madhusudan Tandale, V.I.T., Pune.
2. Ms. Rupali Deshpande, V.I.T., Pune.
3. Mr. Pratul Gadagkar, University of Pune, Pune.
4. Dr. Radha Bhate, University of Pune, Pune.
5. Mr. A. S. Nanajkar, Fergusson College, Pune.
6. Mr. D. N. Sheth, Sir Parashurambhau College, Pune.
7. Ms. Fathia Mohammed Al Samman, University of Pune, Pune.
8. Mr. Rajanish Malekar, National Defence Academy, Pune.
9. Ms. Anuradha Gadre, University of Pune, Pune.
10. Mr. C. S. Nimkar, N. Wadia College, Pune.
11. Mr. Pramod Shinde, N. Wadia College, Pune.
12. Ms. Sunita Patil, University of Pune, Pune.
13. Ms. Nita Kankane, M.I.T., Pune.
14. Mr. Ashok Bhavale, Sinhgad College of Science, Pune.
15. Mr. Hemant R. Pawar, National Defence Academy, Pune.

All participants.

Speakers: Prof. S. A. Katre, Sumit Bharadwaj, Dr. Jayanthan, Niyam Bhushan, Dr. Ajit Kumar, Amitabh Trehan, Manjusha Joshi.

TUG at Bay

Nelson Beebe, Wendy McKay and Ross Moore

Over the extended weekend 16–18 January 2004, some TEX and TUG-related events took place in the San Francisco Bay area. These were:

- visit to Adobe Systems Inc., based in San Jose, to discuss aspects of how Adobe's software interacts with TEX-related workflows, and 'bugs' affecting the onscreen viewing of TEX-typeset PDF documents;
- lunch with the Grand Wizard, at a restaurant in Palo Alto, with a presentation of a drawing by Duane Bibby and shown at TUG 2003 in Hawai'i;
- discussions concerning workflows for the production and archiving of scientific journals, and the preparation/presentation of related bibliographic data on websites of scientific societies and academic institutions.

The first of these included a repeat of a similar meeting 2 years ago, when the "TEXFonts–AdobeApps" (TFAA) working group[1] met with font engineers at Adobe to discuss problems related to fonts, and font rendering, with PDF documents produced using (LA)TEX and pdfTEX. A result of this earlier meeting was the improved rendering, as now implemented in Adobe's Acrobat 6.0 Professional and Adobe Reader 6.0 applications, of PostScript Type 3 (bitmap) fonts. This is extremely important for TEX users, as until only recently this has been primarily the kind of font description produced by `dvips`, at least by default, for TEX and LATEX-typeset documents. Indeed many people continue to use TEX installations where this kind of output is produced.

At that previous meeting in 2002, there were 6 delegates from TUG; this time there were just 3 representatives: Nelson Beebe, Wendy McKay and Ross Moore. But Hans Hagen was there, in spirit, as well. Two days earlier, via a long telephone conversation, Hans had supplied Ross and Wendy with a collection of bugs that he had encountered with the latest Acrobat software. Mostly these were fresh bugs in Acrobat 6, which could be clearly demonstrated by comparison with the output produced by Acrobat 5 displaying the same PDF document.

More than an hour was spent with senior members of Adobe's font development group, discussing general issues as well as presenting the bugs found by Hans. Perhaps the most important result of this meeting is a definite realisation of the need for support of OpenType fonts within TEX software and related applications, such as `dvips` and other driver software. The PostScript Type 1 font technology is no longer the state-of-the-art for fonts. Adobe has not produced any new fonts in Type 1 format for ≈ 6 years; all new development is for OpenType. Thus it is imperative for TEX, pdfTEX and friends to be modified in ways that enable this font format to be used easily.

We discussed the problem of the lack of suitable fonts for mathematics in Adobe's font repertoire, in particular, mathematics fonts that are designed as companions of text font families, such as is the case for Computer Modern, Computer Concrete, Lucida, and MathTime. Regrettably, Adobe sees little or no chance of such support being developed for current font offerings, although in a small number of future fonts, they *may* provide glyphs for the mathematical characters standardized in Unicode.

A whole working day was spent at Adobe, starting at 9.00 am being signed-in by senior Illustrator developer Tom Ruark, also including lunch in the on-site cafeteria, and finishing with the end-of-week beer and nibbles after 5.00pm. Apart from the font issues, most of the day was spent with Tom. The discussions concerned the status of the 'Marked Objects' plug-in [6] for Illustrator, and the bug reports which were specific to Illustrator itself, such as the lack of a consistent set of specifications for the role of the MediaBox, ArtBox, BleedBox, etc., and how these boxes should interact when parts of images are combined. Hans Hagen contributed directly via telephone, as did Gary Gray (U. Pennsylvania). Unfortunately a planned demonstration by Gary could not go ahead, as the 'firewall' would not allow a 'remote-access' connection to be established. Nevertheless, much useful discussion took place, and we departed the Adobe building with the feeling that some understandings had been established and that the reported bugs were likely to be fixed.

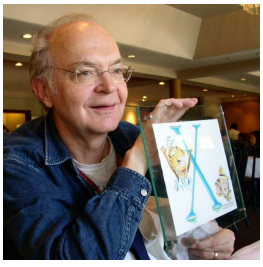


Figure 1: Donald Knuth displays the screen-saver drawing, by Duane Bibby.

[1] See `http://www.tug.org/mailman/listinfo/tfaa` and `http://www.tug.org/twg/tfaa`.

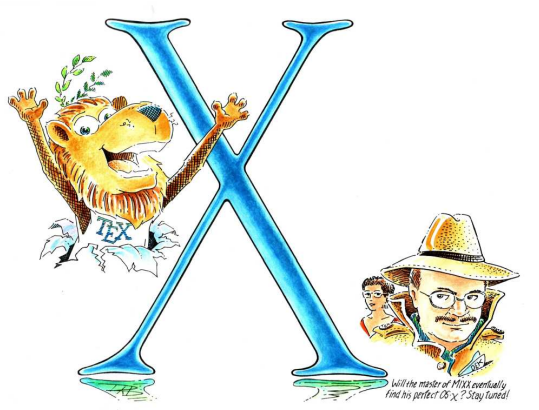


Figure 2: Mac OS X TEX screen-saver, by Duane Bibby.

Figure 3: On the steps of *Ming's* restaurant; from left: Patricia Monohon, Ross Moore, Kaveh Bazargan, Wendy McKay, Donald Knuth, Nelson Beebe.

Later that evening we met with Kaveh Bazargan (Focal Image Inc.) who had flown in from London for a holography conference in San Jose the following week. The main event for the next day, Saturday, was to be lunch with the Grand Wizard of TEX, Donald Knuth. This was at *Ming's* restaurant in Palo Alto. (Unfortunately Jill Knuth could not attend, due to a family matter.) The menu was Chinese 'Dim Sum', with many varied dishes being sampled by all: Donald, Wendy, Nelson, Ross, Kaveh, and Patricia Monohon who had driven up from Ventura, with a car full of memorabilia from the TUG 2003 meeting in Hawai'i, especially for this occasion.

A presentation was made of the original of the Bibby drawing[2] (Figures 1,2) first shown publicly at TUG 2003. Following lunch, photographs were taken on the steps of the restaurant, around a ceremonial lion. (see Figure 3). Some books, purchased just that morning, were autographed. Much of the afternoon was spent driving and exploring the campus at Stanford University, and browsing the University Bookstore.

Sunday saw yet no rest for the wicked. We were joined in Cupertino, where Wendy, Nelson and Ross were staying at a hotel, by Jim Pitman (Department of Mathematics and Department of Statistics, UC Berkeley) who is Chair of the 'Committee on Electronic Issues' of the Institute of Mathematical Statistics (IMS). Kaveh also joined us for a discussion on matters relating to the publication of mathematical journals and archives, both for online access and for paper printing. Central to this discussion was the crisis looming in academia, due to the increasing prices of subscriptions to journals published by the small number of powerful commercial enterprises (see e.g. [7]). That same weekend saw the announcement of the decision by the Triangle Research Libraries Network to cancel their subscriptions to many journals published by Elsevier [4]. Similar cutbacks are being made at other universities [2, 8, 3], as evidenced by a report in the Wall St. Journal [5].

Accordingly, there needs to be greater use and acceptance of online journals, which in turn means that techniques need to be developed that allow for easier management of such resources. One solution is to use a workflow (LA)TEX → XML (for easy storage and archival fidelity) and the reverse XML → LATEX preparatory to typesetting, accompanied by good copy-editing to establish 'clean, standardised LATEX source'. This is the basis of the workflow already used by Focal Image for scientific journals.

These discussions from Sunday continued into the next week, but with a changed venue as Wendy, Nelson and Ross drove up north, to stay on-campus at UC Berkeley. Here it was easier to study various software components developed for creation and management of websites, in particular for bibliographic data. Nelson did a complete installation of his bibliographic software tools [1], for use at the UCB, Department of Statistics site.

[2] This drawing is available in several sizes, to match different computer screens, for general use as a screen-saver or background image, at the TUG 2003 web site: `http://www.tug.org/tug2003/mactex/`.

References

[1] Nelson Beebe. TEX User Group bibliography archive. *University of Utah*, updated daily.

http://www.math.utah.edu/pub/tex/bib/.

[2] Carl T. Bergstrom and Theodore C. Bergstrom. The costs and benefits of library site licenses to academic journals. *Proceedings of the National Academy of Sciences of the United States of America*, January 2004.

[3] Cornell University Library. Links on Scholarly Communication. *Issues in Scholarly Communication*, December 2003. http://www.library.cornell.edu/scholarlycomm/links.html.

[4] From the Hill. UNC Cuts Back on Journals, Cites Burdensome Licensing Agreement. *GAA Online*, January 2004. http://alumni.unc.edu/car/weekly/story.asp?sid=461.

[5] Charles Goldsmith. Reed Elsevier Feels Resistance To Web Pricing. *The Wall Street Journal*, January 2004. http://www.lei.lt/lith3/ivairus/elsevier_feeling.htm.

[6] Wendy McKay, Ross Moore, and Tom Ruark. Adobe plugin for WARMreader. *TUGboat*, 22(3):188–196, September 2001. Talk given at TUG 2001, Delaware. http://tug.org/TUGboat/Articles/tb22-3/tb72moore-warm.pdf

[7] Jim Pitman. The future of IMS journals. *IMS Bulletin*, 32(1), 2003. http://stat-www.berkeley.edu/users/pitman/imsbull.html.

[8] Sidney Verba. A Letter from Sidney Verba. *Harvard University Library*, January 2004. http://www.econ.ucsb.edu/~tedb/Journals/harvardletter040101.htm.

◇ Nelson Beebe
Department of Mathematics, University of Utah
beebe@math.utah.edu

◇ Wendy McKay
Control and Dynamical Systems, California Institute of Technology
wgm@cds.caltech.edu

◇ Ross Moore
Mathematics Department, Macquarie University
ross@maths.mq.edu.au

Institutional Members

American Mathematical Society, *Providence*, *Rhode Island*

Banca d'Italia, *Roma*, *Italy*

Center for Computing Science, *Bowie*, *Maryland*

CNRS - IDRIS, *Orsay*, *France*

CSTUG, *Praha*, *Czech Republic*

Duke University, Vesic Library, *Durham*, *North Carolina*

Florida State University, School of Computational Science and Information Technology, *Tallahassee*, *Florida*

IBM Corporation, T J Watson Research Center, *Yorktown*, *New York*

Institute for Advanced Study, *Princeton*, *New Jersey*

Institute for Defense Analyses, Center for Communications Research, *Princeton*, *New Jersey*

KTH Royal Institute of Technology, *Stockholm*, *Sweden*

Masaryk University, Faculty of Informatics, *Brno*, *Czechoslovakia*

Max Planck Institut für Mathematik, *Bonn*, *Germany*

New York University, Academic Computing Facility, *New York*, *New York*

Princeton University, Department of Mathematics, *Princeton*, *New Jersey*

Siemens Corporate Research, *Princeton*, *New Jersey*

Springer-Verlag Heidelberg, *Heidelberg*, *Germany*

Stanford Linear Accelerator Center (SLAC), *Stanford*, *California*

Stanford University, Computer Science Department, *Stanford*, *California*

Stockholm University, Department of Mathematics, *Stockholm*, *Sweden*

University College, Cork, Computer Centre, *Cork*, *Ireland*

University of Delaware, Computing and Network Services, *Newark*, *Delaware*

Université Laval, *Ste-Foy*, *Québec*, *Canada*

University of Oslo, Institute of Informatics, *Blindern*, *Oslo*, *Norway*

Uppsala University, *Uppsala*, *Sweden*

Vanderbilt University, *Nashville*, *Tennessee*

Calendar

2004

Oct 9	First meeting of GuIT (Gruppo utilizzatori Italiani di TEX), Pisa, Italy. For information, visit http://www.guit.sssup.it/GuITmeeting/2004/2004.en.html.
Oct 18 – 19	Third Annual St. Bride Conference, "Bad Type", London, England. For information, visit http://www.stbride.org/conference.html.
Oct 23	LATEX session at Linux party, Roskilde, Denmark. For information, visit http://www.tug.dk/kalender.html.
Oct 28 – 30	DANTE, 31st meeting, Universität Hannover, Germany. For information, visit http://www.dante.de/events/.
Oct 28 – 30	ACM Symposium on Document Engineering, Milwaukee, Wisconsin. For information, visit http://www.sdml.info/doceng2004/.
Nov 11 – Dec 31	In Flight: A traveling juried exhibition of books by members of the Guild of Book Workers. Boston Public Library, Boston, Massachusetts. Sites and dates are listed at http://palimpsest.stanford.edu/byorg/gbw.
Nov 20	UK TUG Annual general meeting, London. For information, visit http://uk.tug.org/.

2005

Jan 3 – 7	Rare Book School, University of Virginia, January Sessions in New York City. Two one-week courses: The printed book in the West since 1800, and Book illustration processes to 1890. For information, visit http://www.virginia.edu/oldbooks.
Jan 18 – Feb 25	In Flight: A traveling juried exhibition of books by members of the Guild of Book Workers. Scripps College, Claremont, California. Sites and dates are listed at http://palimpsest.stanford.edu/byorg/gbw.
Feb 23 – 25	Seybold Seminars, New York. For information, visit http://www.seybold365.com/2005/.

EuroTEX 2005
Abbaye des Prémontrés (Pont-à-Mousson, France).

Mar 7 – 11	The 15th Annual Meeting of the European TEX Users, and the $2^{2^{2^{2^{2-\infty}}}}$ anniversary of both DANTE and GUTenberg, "Let's TEX Together". For information, visit http://www.gutenberg.eu.org/eurotex2005/.
Mar 10 – Apr 22	In Flight: A traveling juried exhibition of books by members of the Guild of Book Workers. Rochester Institute of Technology, Rochester, New York. Sites and dates are listed at http://palimpsest.stanford.edu/byorg/gbw.
Apr 6 – 8	27th Internationalization and Unicode Conference, "Unicode, Cultural Diversity, and Multilingual Computing". Berlin, Germany. For information, visit http://www.unicode.org/iuc/iuc27/iuc27cfp.html.
Apr 14 – 16	TYPO.GRAPHIC.BEIRUT 2005 Conference, Lebanese American University, Beirut, Lebanon. For information visit http://www.atypi.org/ and look for the entry under "News from members".

Status as of 1 October 2004

For additional information on TUG-sponsored events listed here, contact the TUG office (+1 503 223-9994, fax: +1 503 223-3960, e-mail: office@tug.org). For events sponsored by other organizations, please use the contact address provided.

An updated version of this calendar is online at http://www.tug.org/calendar/.

Additional type-related events are listed in the Typophile calendar, at http://www.icalx.com/html/typophile/month.php?cal=Typophile.

May 22 – 27 Book History at A&M: The Fourth Annual Texas A&M Workshop on the History of Books and Printing. Texas A&M University, College Station, Texas. For information, visit `http://lib-oldweb.tamu.edu/cushing/bookhistory/2005.html`.

Jun 15 – 18 ALLC/ACH-2004, Joint International Conference of the Association for Computers and the Humanities, and Association for Literary and Linguistic Computing, "The International Conference on Humanities Computing and Digital Scholarship", University of Victoria, British Columbia. For information, visit `http://web.uvic.ca/hrd/achallc2005/` or the organization web site at `http://www.ach.org`.

Jun 20 – 23 Seybold Seminars Amsterdam 2005, Netherlands. For information, visit `http://www.seybold365.com/2005/`.

Jul 14 – 17 SHARP Conference (Society for the History of Authorship, Reading and Publishing), "Navigating Texts and Contexts". Dalhousie University, Halifax, Canada For information, visit `http://sharpweb.org/` or `http://www.dal.ca/~sharp05/`.

Jun 6 – Jul 29 Rare Book School, University of Virginia, Charlottesville, Virginia. Many one-week courses on topics concerning typography, bookbinding, calligraphy, printing, electronic texts, and more. For information, visit `http://www.virginia.edu/oldbooks`.

Jul 20 – 24 TypeCon2005, Type Directors Club, New York City. `http://www.tdc.org/news/2004typecon2005.html`

Jul 31 – Aug 4 SIGGRAPH 2005, Los Angeles, California. For information, visit `http://www.siggraph.org/calendar/`.

TUG 2005
Wuhan, China.

Aug 23 – 25 The 26th annual meeting of the TEX Users Group. For information, visit `http://www.tug.org/tug2005/`.

Aug 29 – 31 Seybold Seminars, San Francisco. For information, visit `http://www.seybold365.com/2005/`.

Sep 15 – 18 Association Typographique Internationale (ATypI) annual conference, Helsinki, Finland. For information, visit `http://www.atypi.org/`.

TUG 2005
International Typesetting Conference
Announcement and Call for Papers

TUG 2005 will be held in Wuhan, China from August 23–25, 2005. CTUG (Chinese TEX User Group) has committed to undertake the conference affairs.

Wuhan is close to the birthplace of Taoism and the Three Gorges Reservoir. China is also the birthplace of typography in ancient times, and is simply a very interesting place to go.

For more information, see the conference web page at `http://tug.org/tug2005`, or email `tug2005@tug.org`.

Call for papers

Please submit a title and abstract for papers or presentations by April 1, 2005, via email to `tug2005@tug.org`. Any TEX-related topic will be considered.

Conference fees

The conference fees and deadlines for members of any TEX user group (in US dollars):

Early registration	May 20, 2005	$100
Normal registration	July 1, 2005	$220
Late registration	August 1, 2005	$380

In all cases, non-user group members add $20.

Conference bursary

Some financial assistance is available. The application deadline is March 25, 2005. Please see `http://tug.org/bursary` for details.

Hope to see you there!

Late-Breaking News

Production Notes

Mimi Burbank

It has been quite a while since I've written one of these columns. Our production team (whose names may be found on the back of the title page) has grown to include some new members. Most notable of these has been Karl Berry, who has provided technical input, editorial input, as well as having gone out and "beat the bushes" to obtain new and interesting articles. The major problem over the past several years has been lack of material.

We now electronically transfer PDF files to our printer — all created with pdf(LA)TEX and ConTEXt. The overwhelming majority of production files are in LATEX these days, while the covers and calendar are still produced in plain TEX. Which brings us to the discussion of plain TEX versus LATEX — much of what is done with plain TEX in *TUGboat* dates back to a time when *TUGboat* was produced using plain TEX and followed by more than a few years with only a few LATEX articles. There are things that one can do with plain TEX that cannot be done with LATEX, not because it *can't* be done, but because it is just so much simpler to do it with plain TEX . . . and vice versa!

Since this situation isn't likely to change soon, these production notes may be omitted from future issues unless there is something special.

Future Issues

The next issue will contain the proceedings of the EuroTEX 2003 meeting held in Brest, France. Yannis Haralambous provided the files to the *TUGboat* production team, and we are in the process of finishing up the final editing cycle.

◇ Mimi Burbank
School of Computational Science
Florida State University,
Tallahassee, FL 32306 – 4052
mimi@csit.fsu.edu

TUG financial statements for 2003

Robin Laakso

This financial report for 2003 also includes financial statements from 2002, for purposes of comparison. As usual, the accounts have been reviewed by TUG's accountant but have not been audited.

TUG is funded primarily by annual dues from members. TUG membership declined slightly from 2002 to 2003, by about 2%. Most notable was the decline in joint membership categories. NTG (the Dutch TEX user group) and UK-TUG (United Kingdom) joint memberships combined dropped about 20% from 2002 to 2003. Total membership dues revenue was greater in 2002, however, because an extra $10,000 (2000 and 2001 UK-TUG joint membership dues) was collected in that year.

The TUG Store was launched in April, 2003 (`http://tug.org/store`). Thirty-five copies of the TEX Live software and 24 copies of *TUGboat*, as well as TEX pins, conference mugs and t-shirts were purchased via the store, boosting the "Product Sales" category by $2040.

Notable contributions and allocations made by TUG in 2003:

- TUG 2003: $10000, of which about half was recouped. Also, we thank both numerous individuals and the other user groups, especially DANTE e.V., GUST, NTG, and GUTenberg, for very generous contributions to the conference.
- TEX development fund: $5000
- TEX workshop in Pune, India: $2100
- EuroTEX 2003: $2000
- Bursary fund: $2000

As usual, TUG's largest annual expense items were payroll, *TUGboat* production and mailing, and software production and mailing, all of which decreased slightly in 2003.

The $269 loss in 2003 compared to a positive $7180 net income in 2002 can be explained almost entirely by extra joint member dues received in 2002 in combination with greater contributions made by TUG in 2003. Overall, TUG's financial picture remained quite steady in 2003.

This verbal portion of the report is intended to highlight major features contained in the financial statements, but cannot explain detailed activities within each account. If you would like to learn more about TUG's finances or have a particular comment or question, please contact the TUG office.

◇ Robin Laakso
TUG Executive Director
office@tug.org

TeX Users Group
Balance Sheet Prev Year Comparison
As of December 31, 2003

	Dec 31, 03	Dec 31, 02
ASSETS		
Current Assets		
Checking/Savings		
OregonTelco PrimeShare	133,750	128,258
BOA Maximizer	28,902	51,780
BOA Checking	-7,527	5,730
BOA Money Mkt Bursry	1,711	1,330
Petty Cash	10	10
Total Checking/Savings	156,846	187,108
Accounts Receivable		
Accounts Receivable	300	7,270
Total Accounts Receivable	300	7,270
Other Current Assets		
Deposits	10	
Total Other Current Assets	10	
Total Current Assets	157,156	194,378
Fixed Assets		
Fixed Assets		
Equipment	44,625	44,025
Accumulated Depreciation	-40,300	-36,966
Total Fixed Assets	4,325	7,059
Total Fixed Assets	4,325	7,059
TOTAL ASSETS	**161,481**	**201,437**
LIABILITIES & EQUITY		
Liabilities		
Current Liabilities		
Accounts Payable		
Accounts Payable	31,104	40,124
Total Accounts Payable	31,104	40,124
Other Current Liabilities		
Deferred conference donations	100	610
Deferred conference income		7,360
Deferred contributions		1,500
Deferred member income		21,175
AMS Prepaid Memberships	1,800	1,800
Payroll Liabilities		
Federal P/R Taxes Payable	885	994
State P/R Taxes Payable	195	218
Total Payroll Liabilities	1,080	1,212
Total Other Current Liabilities	2,980	33,657
Total Current Liabilities	34,084	73,781
Total Liabilities	34,084	73,781
Equity		
Restricted DevFund as of 12/31	3,433	
Restricted Bursary as of 12/31	1,711	1,330
Restricted LaTeX3 as of 12/31	-76	168
Unrestricted as of 1/1	122,588	118,979
Net Income	-259	7,180
Total Equity	127,397	127,657
TOTAL LIABILITIES & EQUITY	**161,481**	**201,438**

TeX Users Group
Profit & Loss Prev Year Comparison
January through December 2003

	Jan - Dec 03	Jan - Dec 02
Ordinary Income/Expense		
Income		
Membership Dues	113,597	125,215
Product Sales	4,955	3,050
Contributions Income	5,743	5,065
Annual Conference	4,915	-393
Annual Regional Conference		-363
Conference Classes		-314
Interest Income	6,064	5,130
Advertising Income	400	1,345
Bursary	381	301
TeX Development Fund	3,433	
LaTeX 3	-234	-520
Miscellaneous Income	0	
Total Income	139,254	138,516
Cost of Goods Sold		
TUGboat Prod/Mailing	22,500	24,189
Software Production/Mailing	10,207	13,659
Postage/Delivery - Members	3,684	4,184
TUG Store - shipping	299	
Conf Expense, office + overhead	3,698	
Member Renewal	469	420
Copy/Printing for members	67	60
Total COGS	40,924	42,512
Gross Profit	98,330	96,004
Expense		
Contributions made by TUG	21,100	5,942
Office Overhead	9,318	8,021
Payroll Exp	60,091	60,460
Contract Labor	735	375
Professional Fees	1,651	14,905
Credit card/Bank charges	3,769	3,137
Depreciation Expense	3,334	2,786
Interest Expense		3
Total Expense	99,998	95,629
Net Ordinary Income	-1,668	375
Other Income/Expense		
Other Income		
Prior year adjust (02-03)	-3,592	6,806
Other Income	5,000	
Total Other Income	1,408	6,806
Net Other Income	1,408	6,806
Net Income	**-260**	**7,181**

2005 TEX Users Group Election

Barbara Beeton
for the Elections Committee

The positions of TUG President and of eight members of the Board of Directors will be open as of the 2005 Annual Meeting, which will be held in August 2005 in Wuhan, China.

The current President, Karl Berry, has stated his intention to stand for re-election. The names of the directors whose terms will expire in 2005 are: Steve Grathwohl, Jim Hefferon, Arthur Ogawa, Gerree Pecht, Steve Peter, and Mike Sofka. Two additional director positions are currently unoccupied. Continuing directors, with terms ending in 2007, are: Barbara Beeton, Kaja Christiansen, Susan DeMeritt, Ross Moore, Cheryl Ponchin, Samuel Rhoads, and Philip Taylor.

The election to choose the new President and Board members will be held in Spring of 2005. Nominations for these openings are now invited.

The Bylaws provide that "Any member may be nominated for election to the office of TUG President/to the Board by submitting a nomination petition in accordance with the TUG Election Procedures. Election ... shall be by written mail ballot of the entire membership, carried out in accordance with those same Procedures." The term of President is two years.

The name of any member may be placed in nomination for election to one of the open offices by submission of a petition, signed by two other members in good standing, to the TUG office at least two weeks (14 days) prior to the mailing of ballots. (A candidate's membership dues for 2005 will be expected to be paid by the nomination deadline.) The term of a member of the TUG Board is four years.

A nomination form follows this announcement; forms may also be obtained from the TUG office, or via the TUG Web pages at `http://www.tug.org`.

Along with a nomination form, each candidate must supply a passport-size photograph, a short biography, and a statement of intent to be included with the ballot; the biography and statement of intent together may not exceed 400 words. The deadline for receipt at the TUG office of nomination forms and ballot information is **1 February 2005**.

Ballots will be mailed to all members within 30 days after the close of nominations. Marked ballots must be returned no more than six (6) weeks following the mailing; the exact dates will be noted on the ballots.

Ballots will be counted by a disinterested party not part of the TUG organization. The results of the election should be available by early June, and will be announced in a future issue of *TUGboat* as well as through various TEX-related electronic lists.

2005 TUG Election — Nomination Form

Only TUG members whose dues have been paid for 2005 will be eligible to participate in the election. The signatures of two (2) members in good standing at the time they sign the nomination form are required in addition to that of the nominee. **Type or print** names clearly, using the name by which you are known to TUG. Names that cannot be identified from the TUG membership records will not be accepted as valid.

The undersigned TUG members propose the nomination of:

Name of Nominee: _____

Signature: _____

Date: _____

for the position of (check one):

☐ **TUG President**

☐ **Member of the TUG Board of Directors**

for a term beginning with the 2005 Annual Meeting, **August 2005**.

Members supporting this nomination:

1. _____ (please print)

   _____ (signature) _____ (date)

2. _____ (please print)

   _____ (signature) _____ (date)

Return this nomination form to the TUG office (FAXed forms will be accepted). Nomination forms and all required supplementary material (photograph, biography and personal statement for inclusion on the ballot) must be received in the TUG office no later than **1 February 2005**.[1] It is the responsibility of the candidate to ensure that this deadline is met. Under no circumstances will incomplete applications be accepted.

- ☐ nomination form
- ☐ photograph
- ☐ biography/personal statement

TEX Users Group **FAX:** +1 503 223-3960
Nominations for 2005 Election
1466 NW Naito Parkway, Suite 3141
Portland, OR 97209-2820
U.S.A.

[1] Supplementary material may be sent separately from the form, and supporting signatures need not all appear on one form.

TEX Consultants

Ogawa, Arthur
40453 Cherokee Oaks Drive
Three Rivers, CA 93271-9743
(209) 561-4585
Email: arthur_ogawa@teleport.com
Bookbuilding services, including design, copyedit, art, and composition; color is my speciality. Custom TEX macros and LATEX2ε document classes and packages. Instruction, support, and consultation for workgroups and authors. Application development in LATEX, TEX, SGML, PostScript, Java, and C++. Database and corporate publishing. Extensive references.

Veytsman, Boris
2239 Double Eagle Ct.
Reston, VA 20191
(703) 860-0013
Email: boris@lk.net
I provide training, consulting, software design and implementation for Unix, Perl, SQL, TEX, and LATEX. I have authored several popular packages for LATEX and `latelx2html`. I have contributed to several web-based projects for generating and typesetting reports.
For more information please visit my web page: `http://users.lk.net/ borisv`.

The information here comes from the consultants themselves. We do not include information we know to be false, but we cannot check out any of the information; we are transmitting it to you as it was given to us and do not promise it is correct. Also, this is not an endorsement of the people listed here. We provide this list to enable you to contact service providers and decide for yourself whether to hire one.

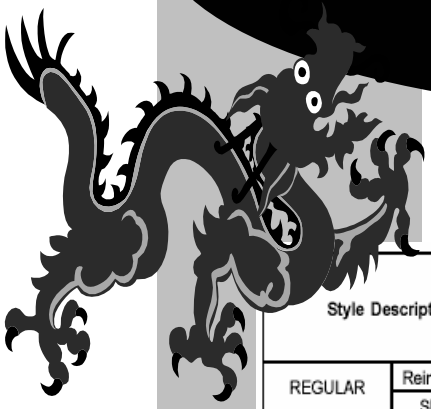
The TUG office mentions the consultants listed here to people seeking TEX workers. If you'd like to be included, or place a larger ad in *TUGboat*, please contact the office or see our web pages:

TEX Users Group
1466 NW Naito Parkway, Suite 3141
Portland, OR 97208-2311, U.S.A.

Phone: +1 503 223-9994
Fax: +1 503 223-3960
Email: office@tug.org
Web: http://tug.org/consultants.html
http://tug.org/TUGboat/advertising.html

EASY TABLE

KHANH HA



A TEX Table Macro Package

Style Description		Compare		Order 6 Pairs of			Save 50% Order 6 Pairs Pay Only
		Reg. Retail Price Per Pair	Sale Price Per Pair	Style No.	Size	Color No.	
REGULAR	Reinforced Toe	$1.99	99¢	3811C	1X, 2X, 3X, or 4X	Suntan (64) Coffee(65) Nude (89) Taupe (16) Off Black (74)	$5.94
	Sheer Toe			3815D			
CONTROL TOP	Reinforced Toe	$2.99	$1.69	3815D			$8.94
	Sheer Toe			3855C			
SILKY SHEER	Sheer Toe	$2.99	$1.69	3852D			$8.94
LIGHT SUPPORT	Reinforced Toe	$3.99	$1.99	3861D			
	Sheer Toe			3865C			

CODES:

```
\tab{&&\bspan[3-4] Compare&&\bspan[5-7] Order 6 Pairs of\et}
\prul[3-7,3pt,1]
\tab{\xyspan[1-2]{ Style Description}&&Reg. Retail Price Per Pair&Sale Price Per Pair&
Style No.& Size& Color No.&\lrow{ Save 50\%\ Order\nl 6 Pairs\nl Pay Only}\et}
\hrul{6pt}{1}
\tab{&Reinforced Toe&&&3^11C\et}
\prul[2-2,3pt,1]\mrul[5-5,3pt,1]
\tab{\xrow{REGULAR}&Sheer Toe&\xrow{\$1.99}&\xrow{99\cent}&3^15D&&&\xrow{\$5.94}\et}
\prul[1-5,3pt,1]\mrul[8-8,3pt,1]
. . .
\tab{SILKY SHEER&Sheer Toe&\$2.99&\$1.69&3^52D&\xrow{1X, 2X, 3X, or 4X}&&\$^.94\et}
\prul[1-5,3pt,1]\mrul[8-8,3pt,1]
\tab{&Reinforced Toe&&&3^61D\et}
\prul[2-2,3pt,1]\mrul[5-5,3pt,1]
\tab{\xrow{LIGHT SUPPORT}&Sheer Toe&\xrow{\$3.99}&\xrow{\$1.99}&3^65C&&
\xrow{Suntan (64)\nl Coffee(65)\nl Nude (89)\nl Taupe (16)\nl Off Black (74)}\et}
```

◊ ◊ ◊

This TEX table macro package rivals the best of the commercial typesetting systems.

Cost? $49.95

- Dynamic table setting by template control
- Multiple mixed column spanners, subspanners, and row spanners
- Easy routines to split table footnotes and break extremely long tables
- Partical hrules, floating hrules anywhere, any length on exact baselineskip
- Automatic decimal alignment, or any special character, in irregular tables
- End columns with \et command anywhere and all vrules are automatically drawn
- Old article: http://tug.org/TUGboat/Articles/tb11-2/tb28ha.pdf
- Version10.04, far superior to the original 1989 version, is now available.

Inquire about EASY TABLE at:
www.authorkhanhha.com/EZ
301-523-4242