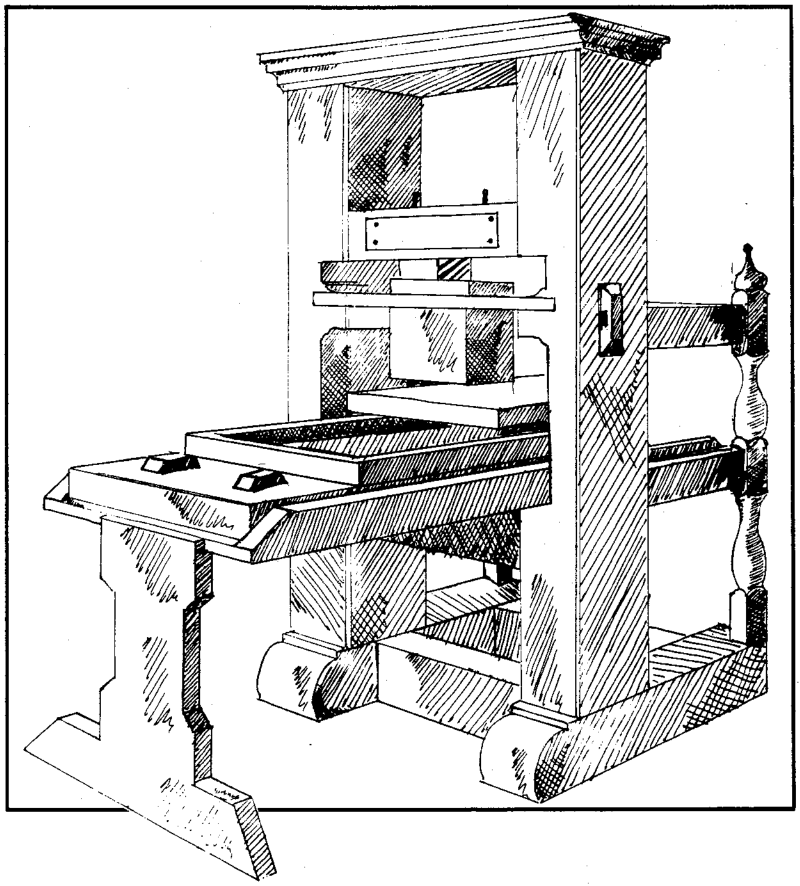
TUGBOAT

The Communications of the TEX Users Group



Volume 14, Number 3, October 1993
1993 Annual Meeting Proceedings

TeX Users Group

Memberships and Subscriptions

TUGboat (ISSN 0896-3207) is published quarterly by the TeX Users Group, Balboa Building, Room 307, 735 State Street, Santa Barbara, CA 93101, U.S.A.

1993 dues for individual members are as follows:

- Ordinary members: $60
- Students: $30

Membership in the TeX Users Group is for the calendar year, and includes all issues of *TUGboat* and *TeX and TUG NEWS* for the year in which membership begins or is renewed. Individual membership is open only to named individuals, and carries with it such rights and responsibilities as voting in the annual election. A membership form is provided on page 361.

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

Second-class postage paid at Santa Barbara, CA, and additional mailing offices. Postmaster: Send address changes to *TUGboat*, TeX Users Group, P. O. Box 869, Santa Barbara, CA 93102-0869, U.S.A.

Institutional Membership

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

Addresses

General correspondence:
TeX Users Group
P. O. Box 869
Santa Barbara,
CA 93102-0869 USA

Payments:
TeX Users Group
P. O. Box 21041
Santa Barbara,
CA 93121-1041 USA

Parcel post,
delivery services:
TeX Users Group
Balboa Building
Room 307
735 State Street
Santa Barbara, CA 93101
USA

Telephone
805-963-1338

Fax
805-963-8358

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

1993 Annual Meeting Proceedings

TeX Users Group

Fourteenth Annual Meeting

Aston University, Birmingham, England, July 26–30, 1993

TUGboat

COMMUNICATIONS OF THE TeX USERS GROUP

TUGboat Editor	Barbara Beeton
Proceedings Editors	Mimi Burbank
	Sebastian Rahtz

Volume 14, Number 3 • October 1993

Providence • Rhode Island • U.S.A.

Editorial and Production notes

This volume includes the majority of papers presented at TUG'93; the exceptions are Richard Kinch's *True TEX: A New Implementation for Windows*, Richard Southall's *Document Design* and Laurent Siebenmann's *The Spacing Around Math*, which were not offered for publication. One paper, Nelson Beebe's *Bibliography Prettyprinting and Syntax Checking*, was too long for the *Proceedings* and will be published in full in the next issue of *TUGboat*.

The following workshops and panel sessions were held (organiser's name in brackets): *LATEX3* (Frank Mittelbach), *Virtual Fonts* (Michael Doob), *Bibliographic Formatting Tools* (David Rhead), *Multilingual TEX* (Yannis Haralambous), *Archives*, *Typographic Issues*, and *Math Font Encodings*. Alan Jeffrey's report on the latter is included in these *Proceedings*.

This volume breaks with *TUGboat* tradition by making almost no use of Donald Knuth's Computer Modern font family. All the papers are set in Lucida Bright, with Lucida Sans, Lucida Typewriter and Lucida New Math used where appropriate, at just below 9pt. The only exceptions are figures in the papers by Daniel Taupin where the macro packages were strongly wedded to CMR-like font names. We hope that *TUGboat* readers will enjoy this changed look, or at least use it as a conversation piece. It is salutary to relate that many of the papers were carefully tuned by their authors to fit the page size and line breaks *if* they were set in Computer Modern — Lucida caused many headaches for the editors for this reason! Readers may also notice that two different sets of hyphenation patterns were used — all papers by non-American authors were set using UK English hyphenation.

All papers were received, edited and reviewed via electronic mail. Four papers — Haralambous, Taupin (twice) and Plaice — needed their own fonts to be built using METAFONT. Just one figure had to be pasted in, for obvious reasons — the example output in Kawaguti's paper. The nine plain TEX and twenty LATEX files were processed on a Sun SPARCstation (the same machine at Aston which hosts the UK TEX Archive, courtesy of Peter Abbott), and PostScript output prepared with Tom Rokicki's *dvips*. The LATEX files used the New Font Selection Scheme, version 2, throughout, and the plain TEX files used a specially-prepared format file with the Lucida fonts specified. The 'Cork' (DC) encoding was used for all text fonts. The output was printed using PK fonts, generated by *ps2pk* from PostScript Type1 sources. The camera-ready copy was printed on a Linotron phototypesetter.

The reviewing and editing process started later in the year than anticipated, but all the authors were very prompt in getting their papers ready for both the preprints and these final *Proceedings*. The burden of editorial work was undertaken primarily by Mimi Burbank (with greatly appreciated help from Bill Burgess), while Sebastian Rahtz handled the implementation and production. An excellent band of anonymous reviewers provided an enormous amount of feedback without which this volume would have been the poorer.

Sebastian Rahtz and Mimi Burbank

Obituary — Yuri Melnichuk

Yuri Melnichuk, a reader in computing mathematics at Lviv Polytechnical Institute, in Ukraine, and one of the participants at the Aston TUG'93 conference and courses, died suddenly of a heart attack while at the University of York on Friday, August 13th.

Yuri was a fairly frequent visitor to York, where he was working on a joint book on number theory with Maurice Dodson. During his time at York he had been introduced to TEX, and with the help of other colleagues in the Mathematics department there had developed this interest. Realising its potential value to the academic community in the Ukraine he had contacted others, with a view to establishing good links between institutions and individuals.

His concerns were not just mathematical: he was also active in ensuring that the British Council and the IEE were in contact with relevant bodies in the Ukraine. Just before he died he was searching the net for a PC version of Ada to take back for computer scientists in his institution.

Last year, he had been instrumental in beginning a TEX users group in Lviv and was beginning to coordinate its activities over the whole country.

He was a dynamic personality, with infectious vision and enthusiastic plans. An excellent and hospitable host, his good humour was matched with determination. His ability to bring people together was a tribute to his vision of cooperation and his own engaging nature. His loss is a blow to the many friends he had made, to his colleagues in the Ukraine, and to the TEX community worldwide.

Malcolm Clark

Opening words

These proceedings will surprise you. They're not done in Computer Modern! Go look! We don't often have the opportunity to see so much "sample" material done in Lucida! Anyone who's done this sort of work on a large collection of papers can appreciate the difficulties this can mean. Sebastian Rahtz and Mimi Burbank, the co-editors, deserve a solid round of applause for their work.

The conference where it all took place? Very interesting, very nice, very surprising (aren't they all, each in their own way?). For me, this was the first time since Cork (1990) that I'd attended a meeting in Europe. Past European meetings (Prague being the latest one) have clearly created a large group of colleagues and friends over the years. And it was evident at Aston.

It was a great pleasure to meet so many people who had been only names and e-mail addresses. And the pleasure was not just mine — it seemed that once again, the meeting was mainly about people meeting people: seeing people from past meetings, renewing friendships, making new ones, exchanging information and macros and ideas. It was a dynamic meeting and the participants were the highlight.

But of course there were the presentations too! Papers and panels and workshops, then the coffee chats and lunch discussions and dinner meetings — and then there were those late hours spent over at that pub on campus, the Sacks of Potatoes. I always leave a TUG meeting with a sore throat!

The papers here are all well worth your time to sit down and read, a bit at a time. For a view of paragraphs that you'll never forget, may I suggest you turn to the paper by Bogusław Jackowski and Marek Ryćko, "TEX from `\indent` to `\par`."

Of particular interest to those concerned about the future of TEX (and we all are, to varying degrees) are two papers, one by Joachim Lammarsch, one from Phil Taylor, which provide some background on NTS (New Typesetting System), seen by many as the next evolution or generation of typesetting for us as TEX users. It's not a question of agreeing with these positions, or that these are the only positions. It *is* a question of having some information which can be discussed and considered by the wider TEX-using community.

And there were special moments as well: the trip to Stratford, to attend a performance of Shakespeare's *King Lear*, was preceded by a special visit to the Royal Shakespeare Library. The curator was rather bemused by this group of some 30 people who were ooh-ing and aah-ing over the characters and design and colour rather than the content. The play itself was simply staggering in its length (3 3/4 hours) and its impact, especially as I was sitting only seven rows from the stage.

I met many of the editors of other newsletters and publications (Karel Horák, Włodek Bzyl, Tomasz Plata-Przechlewski). I met a fellow from Slovenia (Borut Žnidar); in my ignorance, I asked how he could do any computer work with the war around. His reply: "We haven't had war for two years now." Reality is a very hard wall indeed sometimes. Attendees were from everywhere: from Norway to Israel, from Japan to Ukraine.

And I met Yuri Melnichuk, from Lviv, Ukraine. Malcolm Clark has written about his visit to that city in TTN 2,3. At Aston, I encountered a man who was dynamic and almost bursting with enthusiasm not just for TEX, but for the people he'd met at the meeting, the TEX community at large, and the developing group of TEX users back in Ukraine. He was extremely keen on talking about how this potential user group (a meeting at the end of September of their computer society was to see the formal establishment of a Ukrainian user group) could interact and connect with TUG; he had spent a great deal of time thinking of the issue, and had written his ideas down. We were even corresponding via e-mail on the 12th of August. And so it was a total shock to read in my mail on the 15th of August: Yuri had died of a heart attack on the 13th. Malcolm has written a notice, which you will find elsewhere in these proceedings.

What will come of the efforts he had made to start a user group? We don't know. Yannis Haralambous was going to visit Lviv (originally at Yuri's request) to help set up TEX for Ukrainian users, but this has now fallen through. I hope very much that Yuri's enthusiasm will continue, and maybe we'll be able to report good news on this front in TTN or *TUGboat*.

I hope that you enjoy these proceedings, that you will find some new ideas, new macros, new approaches which will intrigue you, and perhaps become part of your own TEX practices. And who knows: *you* may come to next year's meeting in Santa Barbara with some ideas of your own! Enjoy the papers. And think about putting a circle on your calendars for TUG'94! It'll be our 15th anniversary meeting — a significant milestone.

◇ Christina Thiele
President, TEX Users Group

A World-Wide Window on TEX

The 14th Annual TEX Users Group Meeting, Aston University, UK

Abstract

TUG '93 was organised by:

Chairman:	Peter Abbott
Administration:	Maureen Campbell
Programme:	Chris Rowley and Malcolm Clark
Editors:	Sebastian Rahtz and Mimi Burbank
Courses organiser:	Carol Hewlett
Publicity and bursaries:	Philip Taylor
Social programme:	David Murphy
TUGly Telegraph:	Steffen Kernstock

Acknowledgements and thanks

The Committee would like to publicly acknowledge the many contributions made both to the Bursaries fund (which enabled the Committee to offer assisted places to well over a dozen overseas delegates), and to the Social fund (which enabled them to provide hospitality and refreshments to all delegates).

The following individual members and organisations contributed to one or both funds and the Committee would like to express their enormous gratitude.

E. A. Coxon	ArborText
Dave Eckersley	DANTE
Yukitoshi Fukimura	Kaveh Bazargan (Focal Image)
Hitosh Iwamoto	
Timo Knuutila	GUTenberg
Jost Krieger	Nordic TeX
Heinz Kroger	NTG
Frank Mittelbach	TCI Software
Anna Morawska	TEX88
Norman Naugle	TUG
Richard Quint	UK-TuG
Marc Van Leeuwen	Ewart North (UniTeX)
Alan Wetmore	Y&Y Inc.

Conference Programme

Monday July 26th

Tutorials

Introduction to LATEX (what it is and what it is not): *Sebastian Rahtz and Michel Goossens*

Flavours of TEX: a brief tour: *Chris Rowley*

Getting TEX: how to set up and maintain a TEX system for yourself and your friends: *Allan Reese*

Keynotes

TEX from \indent to \par: *Bogusław Jackowski & Marek Ryćko*

The future of TEX and TUG: *Christina Thiele*

A new typesetting system: is it really necessary?: *Joachim Lammarsch*

LATEX2+; announcement: *Frank Mittelbach, Rainer Schöpf and Chris Rowley*

Workshops

LATEX3 (*Rainer Schöpf, Frank Mittelbach & Chris Rowley*)

Virtual fonts (*Michael Doob*)

Tuesday July 27th

Multilingual

Typesetting Catalan texts with TEX: *Gabriel Valiente Feruglio & Robert Fuster*

Russian TEX issues: looking about and outlook: *Irina A Makhovaya*

Language-dependent ligatures: *John Plaice*

A format compilation framework for European languages: *Larry Siebenmann*

Multilingual TEX: Working group report *Yannis Haralambous*

Panel: Multilingual issues

Bibliographic tools

Bibliography prettyprinting and syntax checking: *Nelson Beebe*

LexiTEX: context-sensitive citations for LATEX: *Frank Bennett*

Horizons

An abstract model for tables: *Xinxin Wang & Derick Wood*

Using TeX and METAFONT to build complicated maps: *Daniel Taupin*

Mixing TeX and SGML: a recipe for disaster?: *Peter Flynn*

Workshops

BibTeX (*David Rhead*)

Multilingual (*Yannis Haralambous)*

Wednesday July 28th

Futures

A beginner's Guide to DSSL: *Martin Bryan*

NTS: a future to TeX?: *Philip Taylor*

True TeX: a new implementation for Windows and beyond: *Richard Kinch*

Building a TeX-based multi-window environment adapted to research work: *Michel Lavaud*

A future for TeX: *Roger Hunter*

Thursday July 29th

Fonts

Virtual fonts in a production environment: *Michael Doob & Craig Platt*

The Khmer Script tamed by the Lion (of TeX): *Yannis Haralambous*

Scalable outline fonts: *Berthold Horn*

A PostScript font installer written in TeX: *Alan Jeffrey*

A versatile TeX device driver: *Minato Kawaguti*

Maths

An application of literate programming: creating a format for the Bulletin of Polish TUG: *Włodek Bzyl and Tomek Przechlewski*

The spacing around mathematics: a quick trip to the limits of TeX: *Larry Siebenmann*

The 'A' in LaTeX: *Barbara Beeton & Christina Thiele*

Archives

The comprehensive TeX archive network - CTAN: *George Greenwade*

Panel: Archives and information

Workshops

MakeIndex (*Joachim Schrod*)

Maths fonts encodings (*Barbara Beeton & Frank Mittelbach*)

Friday July 30th

Macros

Syntactic sugar: *Kees van der Laan*

Galleys, space, and automata: *Jonathan Fine*

Sorting in BLUe: *Kees van der Laan*

News from MusicTeX: *Daniel Taupin*

TeX in the Ukraine: report: *Yuri Melnichuk*

Didot

Teaching typography - the Didot project: *Mary Dyson*

Document design: *Richard Southall*

Panel: Typographic issues

The Future of TEX and TUG

Christina A. L. Thiele
5 Homestead Street
Nepean, Ontario
K2E 7N9 Canada
cthiele@ccs.carleton.ca

Abstract

Challenges which face TUG are also those which face all user groups: the question of where TEX will lead us in the next several years as other programs are developed, many with TEX or TEX-like elements behind more user-friendly front ends. What TEX has allowed us to do for many years is now slowly becoming possible in other software. Where will pure TEX be in the future? Will we still be able to use it? In short — What will happen to us?

We must encourage those who are passionately concerned with these future directions. We must support improvements and changes which do not hinder or go against the main guiding principles behind TEX: that it be portable, that it be backwards compatible, and that it produce beautiful documents.

That is our challenge as a user group. To provide the means by which all of us, as users, can learn to adapt to, and benefit from, the changes and refinements which those who do development work are labouring over so intensely. And to provide the means by which we can improve our own skills, our own knowledge in the use of TEX, in order to be proud producers of those beautiful documents.

Introduction

Earlier this year, in June, the NTG, the Dutch user group, held their 5th anniversary meeting. Although I was unable to attend, I did write up a short text which their chairman, Kees van der Laan, read to the audience. That text serves as a starting point for this paper.

The Dutch group were celebrating their *lustrum*, and Kees' own address to the meeting was an account of their history to date: their activities, their successes, and their experiences.

I also wrote a bit about history, TUG's history. We are now in our 14th year. This meeting reflects what we are now: a group of people, with broad interests and diverse applications, where TEX is now truly an international tool for typesetting documents, and exchanging information. People everywhere are using TEX not merely to typeset, but in fact to make sure that their ideas are being clearly understood by colleagues around the globe. It has, in some sense, become a language itself: "People talking TEX".

It sounds odd. But for some, it is in fact the literal truth. For those who attended last year's meeting at Portland (or who read the paper in the proceedings), we were treated to a wonderful display of a TEX application by Raman, a blind graduate student at Cornell. He has a voice synthesizer set up to interpret LATEX code, both document structure and mathematics, into sound. This was never part of the original plan! People are taking TEX "where no-one has gone before", if I may steal a well-known phrase from television.

Who could have foreseen such applications when TEX first turned up on a computer near you? It was supposed to allow us to typeset beautiful documents. It was supposed to finally make sense of maths typesetting. It was supposed to work within an English-language context, with some facilities for non-English material. Graphics weren't part of the equation, although the hooks were there.

And what do we see today? We have achieved beautiful typesetting — and also some absolutely awful typesetting! TEX is still far and away the best means of representing mathematics — and TEX is also appearing behind the scenes in material which has absolutely nothing to do with mathematical or technical typesetting. As for English — well, that was totally blown wide open at the 1989 Stanford meeting and TEX v.3. Indeed, at this conference audience, we had people representing probably 30 languages and from probably as many different countries. And we are all still "talking TEX". This is our *lingua* *franca.* Our means of making this world smaller, of making each person in it closer to us.

Who could have foreseen that 15 years ago? I expect that we are similarly unable to draw a picture of where TeX will lead us 10 years from now.

Challenges which face TUG are also those which face all user groups: the question of where TeX will lead us in the next several years as other programs are developed, many with TeX or TeX-like elements behind more user-friendly front ends. What TeX has allowed *us* to do for many years is now slowly becoming possible in other software. Where will pure TeX be in the future? Will we still be able to use it? In short — What will happen to us?

What is this "Future TeX"?

There has been a considerable amount of time and effort devoted to this issue — the future of TeX — in the past few years. Each TUG meeting has a section on "The Future". There are great discussions and arguments which thrive on various network lists: from `comp.text.tex`, which is a general TeX list, to ones devoted specifically to the future, such as the `nts-l@vm.urz.uni-heidelberg.de`, set up by DANTE, the German-speaking user group.[1]

This year is no different: at this conference we heard about the future in a paper from Joachim Lammarsch, entitled "A New Typesetting System: Is it Really Necessary?". A number of other papers were presented in the "Futures" section. These papers in this proceedings will be read ... or glanced at ... or passed over by our membership.

And that raises the question: do we all need to be aware of the discussions on the future? Should we all be expert in the various threads of argumentation which exist? Must we all be informed TeX users, in short? Perhaps ... perhaps not.

I would argue that the TeX-using community is now so large that there are sufficient numbers of people who *are* passionately interested and concerned about the future of TeX, the program. Sufficient numbers of people with a broad range of opinions, some of which are completely different from those of others, who are capable of discussing these issues, and achieving a judicious balance and compromise on just what the future of TeX — and the future TeX — ought to look like.

Does that mean I'm saying let someone else worry about the issues? No. Not at all. In fact, I would say that we should *all* be making the attempt at understanding the broad outlines of what is being discussed. It's not always easy to read those articles in *TUGboat* — I agree! But as much as possible, we should be reading at least the abstracts, the introductory paragraphs, just so that we are aware of the general ideas. We aren't in school anymore: there are no tests at the end of the year about how much we have understood! We are mainly just users of TeX. But we ought to be making the effort to understand what what some of the problems are, and what some of the solutions may be.

And this brings me to the following: we must encourage those who are passionately concerned with these future directions. We must support improvements and changes which do not hinder or go against the main guiding principles behind TeX: that it be portable, that it be backwards compatible,[2] and that it produce beautiful documents.

But that support should not be blind. We must be aware of what is being proposed, and here I turn to the developers and say to you: you must similarly be supporting the users of TeX by providing information, updates, mini-courses and workshops, on just what is happening, and what you expect to see happening. An absence of understandable communications can only be detrimental: people will avoid what they cannot understand. As TeX users, most of us already avoid learning what things like `\futurelet` mean! We are masters at not dealing with parts of TeX which exceed our competence and understanding. Unless there is sufficient information flowing back to users, they will similarly avoid using new versions of TeX, or even offshoots of TeX-but-named-something-else.

The range of TeX's influence is extremely broad: some of us use TeX only in a small way in order to do our jobs; some of us, however, earn our livings using TeX. It has therefore become a key element in how many people do their work. And when something has become so critical to so many people, great care must be taken to preserve the overall integrity of the program.

We have spent years acquiring our TeXpertise. We will not give that up easily. We will not willingly throw away 5 or 10 years' worth of experience for something totally different. We don't do it now by switching to something like *PageMaker* or *Word*; why should we leave TeX for something that's different but TeX-like?

The challenge, as I see it, for developers, both commercial and otherwise, is to produce software

[1] Phil Taylor, "The Future of TeX," *TUGboat* 13, **4**, pp. 433-442, December 1992. See also Phil's paper elsewhere in these Proceedings.

[2] I'd like to thank Phil, whose paper here reminded me of this critical point.

that allows both the experienced TEX user to continue applying that experience, and the new user to more quickly, more easily, produce those beautiful documents. That one can address both the overt user of TEX code, and the person who would prefer to not know what's under the hood.

Can TEX be this tool-for-all-users in the future? I think it can. TEX has proven that, with its own flexiblity, and the ingenuity of its users, it can be turned to almost any task. TEX is an incredible achievement, stemming from the work of one man and a group of dedicated and devoted graduate students. It is an extraordinary piece of luck which brought it all into being. Most of us are aware of this past, and feel that the magic must somehow be preserved.

Simply put, TEX is not like any other software program. Granted, it has its gaps, its less-than-ideal elements, but overall, it has a unique place in software history. Our goal must be to maintain that place, and not allow TEX to become a non-portable, non-compatible, fragmented program. Otherwise, it's not TEX!

The Future of User Groups ...

And where do we, as a user group, fit into this future? Because TEX doesn't exist or function in a vacuum. It lives because we are using it. We may have only a passing acquaintance with much of its inner workings, or only wish to have a passing acquaintance with all the discussions about the future of TEX and a future TEX and all that. But we are also a major part of the future of TEX. There is no question about that.

And for users like us — we must continue to improve our own skills, our own knowledge in the use of TEX, in order to be proud producers of the beautiful documents, which are the reason we use TEX.

Because it is not TEX alone which will do this. No program will produce beautiful documents all by itself. It is the *users* of those programs who produce beautiful documents. TEX is merely a tool — an extraordinary tool, certainly — but we are the craftsmen who must learn to use it to its full potential.

Our user groups can help in teaching us how to refine our use, show us ways of improving our knowledge and skills. Our user groups are there to bind TEX users together, to bring us into contact with these better methods and improved skills.

And it is our users who are ultimately the source of that information: the user group is merely the means by which that information can be distributed widely and thoroughly.

That is our challenge as user groups: to be the highway for this information interchange. To provide the means by which those with wide-ranging knowledge and skills can pass their expertise along to others. And to provide the means by which all of us, as users, can learn to adapt to, and benefit from, the changes and refinements which those who do development work are labouring over so intensely.

... and of TUG, Specifically

TEX itself is just a bit older than TUG, by a few years. So let's say it's been out there for about 15 years or so. While a very small portion of users are members of a user group, ours or others, the actual number of users is probably in the hundreds of thousands. Universities are probably still the spawning ground for most of those users, who move with TEX through their research papers and their theses, eventually becoming employed in companies where they either find TEX already in place, or they introduce it, spreading its use even further. There are the academics and scientists themselves who travel from place to place, bringing their research files with them, and invariably TEX is part of that baggage.

And yet — TUG only counts some 2,500 members. If one puts all the other user groups together, they would also probably only account for another 3,000 to 3,500 members. It would seem, then, that our future as a user group should be of more immediate concern to us. How can we attract new members? What about former members? What is our function?

In 1991, I was a member of the committee to devise a mission statement, a short text which would provide some key points of focus for TUG. At the annual meeting that year, held just outside Boston, the Board refined that text. The final mission statement, comprising three main elements, now appears on the inside front cover of all issues of our newsletter, *TEX and TUG NEWS*:

> The TEX Users Group (TUG) provides leadership:
>
> 1. *to encourage and expand the use of TEX,* METAFONT, *and related systems*
> 2. *to ensure the integrity and portability of TEX,* METAFONT, *and related systems*
> 3. *to foster innovation in high-quality electronic document preparation*

The first point, about encouraging and expanding the use of TEX, METAFONT, and related systems, is directly related to the every-day activities of our user group.

The second point, about TEX's integrity and portability, concerns discussions regarding the future of TEX and the shape of a future TEX.

The third point, about fostering innovation, is one which our publications, *TUGboat* and now *TEX and TUG NEWS* (TTN), are in part addressing.

The job which I see before us, in TUG, is to find ways to implement these points. But what ways?

For many, TUG is an organisation which publishes *TUGboat*, and has an annual meeting. Last year, a quarterly newsletter, TTN, was added. Still, for the vast majority of TUG members, these seem to be the main features of the user group, with the publications being the main benefit of membership. For a smaller group of people, those who have called the TUG Office, formerly in Providence, now in Santa Barbara, TUG also has meant a phone number and a person who might be able to answer their questions, or pass them on to someone else for an answer. It has been a place where one could buy TEX-ware and TEX publications.

But TUG has become more than that. With the advent of the Technical Council (TC), and its various Technical Working Groups (TWGs), TUG is now also showing some of that leadership which our mission statement would have us demonstrate.[3]

Some of the TWGs have become extremely active and productive. Two of them were active at this conference: Yannis Haralambous, chair of the Multiple Language Coordination TWG, led a meeting and George Greenwade, chair of the TEX Archive Guidelines TWG, introduced the Comprehensive TEX Archive Network for the first time.

Other TWGs include one on TEX Extended Mathematic Font Encoding, chaired by Barbara Beeton; Raman, mentioned earlier in this paper, is chair of a TWG on TEX for the Disabled; and there is also a System Implementor Coordination TWG, chaired by Michael Ferguson. The TC, with Ferguson as chair, has also put out a call, published in the latest issue of TTN, for "Special Interest Groups."

The work being done by all these working groups — all of it volunteer, it must be emphasised — is in progress. Some groups are already at the point where they can say their first tasks have been completed; some are right in the middle of making their recommendations and decisions; others are still gathering information. But the point is: the Technical Working Groups are providing *all* TEX users with solid effort and results, which will enhance our use of TEX, our access to TEX, and which will bring some standardisation and guidelines to the TEX-using community.

[3] A basic outline of the Technical Council and its working groups can be found in TTN 1,3:5–8.

Another area of group activity within TUG is our committees. While it could be said that these have been less dynamic or productive in the past, our current committees are now very effective. We will continue this committee approach to addressing issues which face us, as a user group. I would very much like to see TUG become more decisive, more active, with effective and pragmatic mechanisms in place, so that we can become the TUG that is described in our mission statement: demonstrating leadership in those three areas of focus.

By leadership, I certainly don't mean dominance. Leadership, when we drafted the mission statement, was intended to reassert the place of TUG, the TEX Users Group, as the oldest user group, the one which began with the core of people involved with that first developer, Don Knuth. We are the *international* user group, and therefore it is expected that we should demonstrate leadership, that we should support and disseminate information about TEX and TEX-related developments, that where there is a question, somewhere within TUG's collective knowledge lies the person, or the people, with the answer.

We have not always lived up to that expectation; I think we should make every effort to do so.

We are currently reviewing our internal structure, with respect to representatives from other user groups. Since 1989, we have had representatives from five user groups; we are working to come up with formal mechanisms to include other user groups, to clarify the role of such representatives, to ensure that there is open two-way communications between TUG and other groups.

This is an important issue, even if mainly administrative: TEX is now a completely internationalised program. TUG's membership directory reads like a world atlas. The people who have come to this meeting are also proof of the international fact: we have people from Russia, from Spain, from Japan, as well as North America. If we, as the international user group, are to function in harmony, then we must have comments and opinions from everyone. And this means providing a forum for such comment and opinion to be heard.

One big feature of our user groups is the extent to which people are volunteering their time, and their work, to TEX. There are people who volunteer their programs, their macros, their expertise, their information. There are people who volunteer their time to organise meetings, to write newsletters, to write articles. There are people who volunteer to work in groups, TWGs or committees or other names, to try and solve a problem, provide recommendations, establish standards. The volunteer effort which people make to TEX, and usually without a second thought, is extraordinary. We should remember to be thankful for such generosity.

Another feature of our user groups is electronic mail. I think it's safe to say now that the majority of TEX users are also network e-mail users. We communicate as easily with Japan as with our colleagues across the hall (occasionally even better!). We are able to work as a team, discuss our options, disagree and then come to an agreement, and maybe, at some future annual meeting, we may actually meet one another. The cooperation which makes this volunteer effort worthwhile is simply astounding. Added to this is the fact that we are probably, at one time or another, working with colleagues for whom English is not the first language. And yet it still seems to work.

Of course, it's not all roses out there. There have been some terrific battles (also known as "flame wars") waged via e-mail; it is a medium which can foster tremendous misunderstandings, can be perceived as very hurtful and rude, can allow one to make too hasty a reply, when some time spent cooling off might be more useful, and more effective.

The point I'm making here is that, for all the difficulties which it sometimes appears we are suffering, as TEX users, this is a large volunteer-based community. And while we may want to see things move faster, see services improved, get more for our money, and so on, it will really only happen with the cooperation and participation of volunteers, who are people just like you!

So what is the future plan for TUG? I think it involves increasing our membership; increasing and improving our services to members and non-members; serving as that channel, that highway for information exchange. I think our future lies in providing documentation and software on TEX, METAFONT, and related systems.

But more than any of these activities, the future of TUG lies with its members. When members don't renew, we have to ask why. When TEX users choose not to join TUG, or any other user group, we must ask why. When members don't participate in meetings, in elections, in writing for our publications, we must ask why!

There are elections coming up this fall, for all 15 positions on the Board. These will be our second elections. This is one of the most direct ways you have to participate in your user group. Time now for you, as TUG members, to look at those you've elected, to look at what's been accomplished. Election materials will be mailed to all members in the fall. Read carefully, and mail in your ballots.

There are others ways, too, in which you can participate. If you have a particular expertise, an area where you feel you have a certain competence, why not consider letting the TUG Office know that you'd be willing to respond to queries in that area. Even if you volunteer for a 3- or 6-month period, that participation will benefit many other users. I have fielded a number of queries over the years regarding phonetics and linguistics material; I volunteer to continue that effort. If someone wants to join me, that would be wonderful!

Another idea to try out: in German, there's a word — "Stammtisch". It means, more or less, "a regular table set aside for a group". You may have noticed this word appearing in *TUGboat*'s calendar of events. What's happening is a group of people get together on a regular basis, usually in a pub or restaurant, and talk — mainly — about TEX. It's a way of letting TEX users know there's a place where they can find some friends, some fellow users, and talk shop. Or maybe discuss something completely different, like politics, or the weather! But it's an interesting idea which could certainly work anywhere there are TEX users. Why not give it a try? You might be surprised at the results.

And of course, you can participate by talking about TEX whenever the opportunity arises. Recently, I attended a conference for scholarly publishers. Some had heard of TEX; some were actually long-time users. But they had never seen any actual TUG representatives at their meetings — they got two at this one! Peter Flynn, our Secretary, was also able to come to one of the afternoon sessions.

Find out what conferences are going on locally! Many of us belong to other societies; why not send information on their conferences to the TUG Office, if it seems there is a connection with publishing, with typesetting, or with electronic communications. Let us know what's happening in your part of the world. Maybe there's a way we can introduce a TEX presence, either directly via TUG, or via a local user group.

And finally: TUG a member! Bring in an additional member to TUG, and we can discuss a mug or t-shirt for you!

A New Typesetting System: Is It Really Necessary?

Joachim Lammarsch
Computing Center
Universität Heidelberg
Im Neuenheiner Feld 293
D6900 Heidelberg 1
Germany
Internet: X92@vm.hd-net.uni-heidelberg.de

Abstract

It was difficult for me to decide how to prepare this talk and in which function I should give it — as an individual, as a member of the University of Heidelberg, as a Special Director of the TUG Board of Directors (there was already a talk scheduled to be given by Christina Thiele as President of TUG), or as the president of DANTE e.V. I decided to give my talk as the president of DANTE.

NTS — is it really necessary? The answer is obvious — YES!

There are two reasons why NTS is necessary; on the one hand there are technical reasons and on the other, political reasons. Concerning the technical aspects, there were many things to be taken into account: Frank Mittelbach and David Salomon proposed a lot of design changes to Donald Knuth, and there were an increasing number of demands for the TEX Users Group to become active. I do not wish to discuss technical issues because Phil Taylor will give a talk on this subject and he is far better qualified to do the job. My talk is concerned with political reasons for NTS. I will try to show why the NTS project is necessary and what has been done over the last two years, especially during the last few months.

We need the NTS-project and it is easier to understand this need if we have a closer look at the TEX world. First, we have the TEX Users Group, which was founded more than fourteen years ago; it claims to be the international TEX organization and currently has about 2,500 members. Besides TUG, we have the following organizations in Europe (in alphabetical order): DANTE e.V., GUTenberg, the Nordic Group, NTG, UKTEXUG, and now several groups in Eastern Europe. I think these groups together have more members than the TEX Users Group. Most of these members are not members of TUG, but they use TEX too. It is not known how many people currently use TEX worldwide, and it is impossible to estimate how many people would use TEX if it was more user friendly. We can say that the TEX Users Group provides very good support in North America, but I think this support has not been really satisfactory for Europeans. Also, the membership fee for the TEX Users Group is especially high for members of Eastern Europe.

Unfortunately, we also have problems in Europe; for instance, the problem of different languages. There was some support from Donald Knuth to make TEX more flexible, but the solution was very difficult. At the 1990 meeting in Cork, we decided that we needed a standard for the now-called DC-fonts (I even named it "Cork Standard"). The "Cork Standard" is now under discussion. It is not actually a standard anymore and it is obvious that it is controversial.

We tried to establish a European TEX Organization (ETO), but we ran into the same problems as big politics. And a small problem became a big problem: a common language. We have different languages in Europe and it was not easy to agree on a common language. I am glad this type of problem does not occur with local members. But we have one thing in common worldwide, we have a joining element: TEX. And I think the important point is that we never forget this.

We have tried to make TEX attractive for the TEX community and the TEX members, and all have worked together: journals like "*TUGboat*" and "*TEX and TUG NEWS*", or journals for local users in different languages such as "Die TEXnische Komödie" or "Cahiers GUTenberg", the journal of NTG and a lot of other journals. Sometimes these are difficult to read, for me, because they are in Czech or Polish or ..., but that's Europe! We have worldwide discussion lists like INFO-TEX or in Europe TEX-EURO, and we have local lists for local groups: TEX-D-L, GUT, TEX-NL, ..., we have lists for special items, and we have digests like TEXhax and UKTEX. We have user groups and we have many possibilities for communication; but, do we really use these opportunities?

We have a very good software distribution. The distribution is done by the TEX Users Group, by local groups and by the new Comprehensive TEX Archive Network (CTAN). CTAN is a TUG-sponsored project, under the Technical Council, and represents the work of George Greenwade, Sebastian Rahtz and others and the management of the three largest software servers in the world — the TEX network. I think it is the biggest service for any kind of software worldwide.

But we have a big problem: TEX is getting out of date. TEX is now more than 15 years old. I say that ten years ago, it was the best typesetting system worldwide. Today it is the grandfather of the typesetting systems; other typesetting systems have learned from TEX. Other typesetting systems often incorporate many features of TEX, like the hyphenation algorithm, the page-breaking algorithm, kerning, ligatures, and many other things. We find functions for typesetting mathematical formulas in these other systems. TEX is free and commercial vendors have had the chance to use TEX algorithms. There are other systems in use today: Word, WinWord, WordPerfect, Framemaker and others. I do not believe they are as good as TEX, but almost. They provide a better user interface — which is really important — and they do one thing which we will never be able to do: they do advertising! We have never had the money to advertise.

What will happen? The result is that more and more users are deciding to leave the TEX world and start using other systems. There are a lot of people who do not consider the advantages of using TEX. So, we lose users and that means we lose members. This trend has been painfully obvious to user groups — especially the TEX Users Group, which is the oldest. You can see the decreasing number of TUG members and I believe that local groups will begin to suffer the same decrease. Decreasing membership means a decreasing income, which leads to financial problems. If we have financial problems, the services we offer decrease; the members get angry and do not renew their membership. What is the result? Again, a decreasing number of members, decreasing income and even worse service due to financial problems. A vicious cycle! And in the end we have no user groups.

What can we do? The answer is very simple: improve TEX! Make TEX more user friendly! Make TEX more attractive! This is not possible because TEX is frozen by Donald Knuth. We have to live with TEX in its present state, and this is not the state we want.

The consequences are obvious. We need a new typesetting system. This is easy to say, but there is no Donald Knuth to do the work and so therefore we need *all* the people who are able to do such a work. Together we have a chance to create such a new typesetting system. It is also very important that the development be done worldwide, because only a worldwide system is acceptable as the successor of TEX. TEX is the same worldwide. When I write a text in Germany and send it to the US, I can be sure that if the text is printed, it will produce the same output. I know for a fact that a lot of mathematicians use TEX only for communication to send mathematical formulas back and forth from Germany to the US because it is the only way to send formulas via electronic mail. TEX is the translator.

The next issue is what to call the new system; its name cannot be TEX because TEX was named by Donald Knuth and he has announced that he is not willing to give this name away. The AMS is holder of the TEX trademark. They feel obliged to respect Knuth's decision that any program which does not meet his requirements can be called simply "TEX". The work has to be done without Donald Knuth because for him the story of TEX is over.

How to start? What have we done? Now a little bit of history. After the Cork meeting it was obvious that nothing would be done by the TEX Users Group because the TUG board was under reconstruction. I was glad that I — as the president of DANTE e.V. — had the chance to initiate the development of such a new typesetting system. I spent quite some time considering how to proceed, and decided to announce the idea worldwide using the different communication lists. It was never planned as a project of DANTE e.V., nor as a project which would be under the control of any other group. I feel no specific group has the ability to direct such a project. If we finally decide that we need a "board" to direct the project, it must be one which represents all existing groups — all existing TEX users worldwide.

The announcement was a huge success. I received lots of mail from all over the world containing many good ideas and announcements that people would like to help. Unfortunately, there was no official statement from any existing user group, suggesting that it did not seem important enough to a user's group. At the 1992 general meeting in Hamburg, the members of DANTE e.V. voted to support the project. This is important because this project needs money. And I needed the members of DANTE e.V. to agree to provide some money.

We established a moderated discussion group and called it NTS-L. The aim of this list was to discuss all aspects of the new project. List subscribers have seen a lot of good ideas, a lot of nonsense, and a lot of things that have no chance of becoming part of the new typesetting system — in other words, a lot of interesting things! We reviewed these ideas in a group headed by Dr. Rainer Schöpf, a process which took nearly nine months.

But we had problems too. Dr. Schöpf was engaged in another project and did not have enough time to fulfill his job as a technical leader. Searching for a solution, we found Phil Taylor and I was very glad that such an expert in TEX was willing to act as the new technical director. Please refer to his paper in these Proceedings.

What will happen in the future? I do not know. There are some questions, some of which I believe are really important.

One of the questions was what company would be interested in the new typesetting system? Before the 1992 meeting in Portland, I had spent some time in the US and had visited some companies. I spoke to Lance Carnes, president of Personal TEX, and with Doug Garnett from Blue Sky Research, and it became clear that they did not like the idea. They have developed their software, and a new typesetting system would force them to develop new software. On the other hand, it was clear to all of them (and I think David Kellerman, the president of Northlake Software, said it very precisely) that we have two possibilities: we can either support TEX or we can leave the TEX world. And that is true. It was felt that we would get support from companies, but not at this time, because a system 'under consideration' is not a thing a company will support, but later.

The next question was what publisher would support it? Before I started the project, it was clear to me that we also would need support from publishers. I had talks with Addison-Wesley (Germany), Springer Verlag (Germany) and International Thomson Publishing (Germany), and they all promised support because they saw that it would be necessary for something to be done in this direction.

There was another question that I knew would come up. What would Donald Knuth say about this project? I visited Donald Knuth in San Francisco before the meeting in Portland in 1992, and spent some hours talking with him about the project.[1] I explained why we need this project and that we did not have the intention of destroying TEX. He understood that the project was necessary and he provided a lot of hints on how to set up such a project. On the other hand, he said that, for him, TEX is enough. For his use, TEX does all he needs. He had written a typesetting system for his books and the typesetting is done by TEX in a wonderful way. He said very clearly that the typesetting project was finished for him.

[1] The meeting was arranged by Peter S. Gordon from Addison-Wesley Publishing Company.

What was new for me was that he did not write TEX alone, but there had been a group of students supporting him. He wrote most of the code of the program himself, but he always had people to discuss difficulties. He thought it was a good idea that we were a group and said it was possible to have a group for such a project. He gave his best wishes. He did not say he was not interested, but he would not work for it.

Some technical questions now arise. Will it be free? It will be freeware for non-commercial use, just like TEX. Commercial use has to be discussed — there is no consideration in this direction at this time. Who will pay for it? (Very important question!) DANTE e.V. will fund the start of the project. Perhaps other groups will decide to support the project financially. In Europe we may have an opportunity to get support from the European Community; we can try to develop this project as an EC project. Perhaps we can get support from companies: publishing companies, software companies. I think we have enough money for the first steps.

The last questions are: who needs the system? Who will use it? I think *we* will use the system, because for everyone who says that TEX is not enough, it is the only chance we have. We need this system for the remaining years of the 20th century and possibly for the beginning of the next. Or, you could say, we all need the system if we are interested in keeping our community alive.

Now we come to the end and this is the beginning: NTS — is it really necessary? I will say it again: YES!

Acknowledgements. First of all I would like to thank the unknown person who made the announcement that I would give a talk here. It was a big surprise to me to read that the president of DANTE e.V. would be one of the speakers. I asked myself, "Do we have a second president?" Well, after some consideration I decided that it would be a good idea to give the talk. I needed help in preparing the talk because I did not have enough time to do it on my own. I would like to thank Luzia Dietsche, for preparing the transparencies and Volker Schaa for proofreading them. I had some problems in receiving a reimbursement for the costs of the meeting because I

was invited for the talk, and it is uncommon in Germany that your employer pays in such cases. So I thank Dr. Peter Sandner, the director of my Computing Center, for funding my journey. My last thanks go to Malcolm Clark and Chris Rowley, who gave me some hints on how to give a talk and I think that I made use of these hints. The most important item was to speak slowly, so that all non-native English speakers would be able to understand me. Because I am not a native speaker either, it was no problem for me to speak slowly!

TEX from \indent to \par

Marek Ryćko
Wydawnictwo Do
ul. Filtrowa 1
00-611 Warszawa, Poland

Bogusław Jackowski
ul. Tatrzańska 6/1
80-331 Gdańsk, Poland

Abstract

A proper answer to even apparently simple questions about TEX can only be answered with a detailed formal specification of TEX's mechanisms, which will allow us to derive the behaviour of TEX in more complex situations.

Introduction

There are some seemingly simple questions about TEX which may be difficult to answer without precise knowledge of TEX mechanisms.

In the following section we will ask three such questions, encouraging the reader to answer them without reading the explanation.

Actually, the explanation follows immediately from a detailed specification of TEX's action at the beginning and at the end of a paragraph. We believe that if such a specification of all TEX's mechanisms existed, answers to most questions concerning behaviour of TEX would be equally simple.

The pivotal sections are 'Switching from Vertical to Horizontal Mode' and 'Switching from Horizontal to Vertical Mode'. The section 'From Input Characters to Commands' contains necessary introductory material.

Questions

In all questions we assume the normal meaning of tokens of plain TEX.

Q1. What is the difference between:
(∗) `\everypar{\def\indent{1}}`
`\indent 3 is a prime number.`
and
(∗∗) `\everypar{\def\vrule{1}}`
`\vrule 3 is a prime number.`
What is typeset in both cases and why?

Q2. Assuming that TEX is in vertical mode, what is the difference between:
(∗) `\parindent=0mm \indent\par`
and
(∗∗) `\noindent\par`
What is appended to the main vertical list and why?

Q3. What is the difference between:
(∗) `\par`
and
(∗∗) `{\par}`
What is the state of TEX after executing these commands in both cases and why?

From Input Characters to Commands

Let us start with a closer look into TEX's way of processing input data. Three levels of the processing can be distinguished:

L1. Reading characters from the input file and transforming them into tokens (lexical analysis).
L2. Expanding tokens.
L3. Executing commands; at this level TEX creates internal lists (horizontal, vertical and math lists), transforms them into boxes and writes some boxes to the DVI file (using the `\shipout` command).

Knuth talks about "eyes," "mouth" and "stomach" of TEX, etc.; we prefer to speak about "levels."

Names and meanings of tokens. In order to understand what happens at the beginning and at the end of a paragraph it is essential to be aware of the difference between names and meanings of tokens.

Following Knuth, we will denote by *`\xyz` the meaning of the command `\xyz` at the beginning of the TEX job. By $\boxed{\texttt{xyz}}$ we will denote a token, the name of which consists of the letters 'xyz'. Such a token is created by TEX from the sequence of letters 'xyz' preceded by a current escape character, usually backslash.

For example, the token $\boxed{\texttt{hbox}}$, the *name* of which consists of the letters 'hbox,' has initially the meaning *\hbox. Saying '\let\hbox=\par' a user may change the meaning of $\boxed{\texttt{hbox}}$ to the current meaning of $\boxed{\texttt{par}}$, most likely to *\par. Incidentally, TEX replaces every empty input line with the token $\boxed{\texttt{par}}$ regardless of the meaning of this token. The meaning of $\boxed{\texttt{par}}$ may be *\par, but $\boxed{\texttt{par}}$ may be also, for example, a macro expanding to a sequence of tokens.

Transforming input characters into tokens. From the point of view of TEX, the input file is a sequence of characters organized into lines. TEX reads such characters one by one and transforms them at level 1 into so-called tokens, according to definite rules. For example, the following sequence of 15 input characters:

    \ e n s p a c e ␣ D o n . . .

is transformed into a sequence of 7 tokens:

    $\boxed{\texttt{enspace}}$Don...

The first one is a control sequence token and the remaining are character tokens stored by TEX along with their category codes.

Each token created at this level is associated with its current meaning which can be either *a primitive meaning* (a meaning that is built into TEX) or it can be *a macro* (a meaning that can be defined by a user in terms of other meanings). Regarding the meaning we can classify all tokens as follows:

(a) with respect to expandability as *expandable* and *unexpandable*;
(b) with respect to primitivity as *primitive* and *macros*.

The expandable tokens can be primitive, like \if, \the, \noexpand, \csname, or they can be macros defined using \def or a related assignment (\edef, \gdef, \xdef).

All unexpandable tokens are primitive. This group contains, among others: tokens like \hskip, \hbox, etc.; letters and other characters; all tokens defined by the \chardef assignment; some tokens defined by \let or \futurelet.

Expanding tokens. Level 2 of TEX, i.e., the expansion level, reads tokens from the input token list and expands them. If the first token in the input token list is expandable, level 2 of TEX expands it, that is, replaces this token (possibly with some tokens following it) with another sequence of tokens.

If — after the replacement — the first token is still expandable, the expansion is repeated until the list starts from an unexpandable token. Obviously, this process may loop infinitely.

For example, the result of expansion of the first token in the input token list:

    $\boxed{\texttt{enspace}}$Don...

is the sequence of tokens:

    $\boxed{\texttt{kern}}$.5em␣Don...

because the first token $\boxed{\texttt{enspace}}$ is expandable (it is a plain TEX macro) and its expansion is '\kern.5em␣'. The token $\boxed{\texttt{kern}}$ is unexpandable, hence no further expansion takes place.

The input token list with an unexpandable token at the beginning is submitted to level 3 of TEX.

Commands. By a *command* we mean an unexpandable (primitive) token at the beginning of the input token list. If a command may or must have arguments, only the first token is a command. For example, in the input token list:

    $\boxed{\texttt{kern}}$.5em␣Don...

the token $\boxed{\texttt{kern}}$ is the command and the tokens '.5em␣' are arguments. They are being read as a part of the process of executing the command.

In general, a command can read arguments from an input list either demanding expansion from level 2 or not.

Level 3 of TEX — the level that executes commands — is the central level. Every time this level is about to execute the next command it "asks" level 2 to prepare the input token list such that at the beginning of the list there is a primitive (unexpandable) token. In turn, level 2 "asks" level 1 for preparing necessary tokens.

Level 3 executes the command according to its meaning, taking into account the current internal state of TEX, including the values of various parameters, and, in particular, taking into account current TEX's *mode*.

One of the results of executing commands is creation of various kinds of internal lists. The types of lists include: horizontal, vertical and math lists.

At every moment TEX is in one of the following six modes determining what type of list it is currently constructing:

(a) vertical mode (v-mode)
(b) internal vertical mode (iv-mode)
(c) horizontal mode (h-mode)
(d) restricted horizontal mode (rh-mode)
(e) math mode
(f) display math mode

At the very beginning of a job TEX is in v-mode and all the lists are empty. A list is constructed by appending new elements to it. The process of list construction can be briefly summarized as follows: mathematical lists are converted into h-lists; an h-list created in h-mode (material for a paragraph) is converted into a v-list and appended to a current v-list; a vertical list created in v-mode is converted to boxes by a page builder; eventually, boxes to which a command `\shipout` is applied are written to a DVI file.

Summary of Paragraph Construction

In the process of creating a paragraph by TEX there are three distinct phases:

P1. Switching from v-mode to h-mode (opening a new h-list — see the section 'Switching from Vertical to Horizontal Mode').

P2. Creating the h-list. (We do not discuss this phase in the paper. The notion of h-list is explained in "The TEXbook," pp. 94-95. The systematic description of how the commands processed in h-mode influence the state of the h-list contain chapters 24 and 25 of "The TEXbook," pp. 267-287).

P3. Switching from h-mode to v-mode (converting the h-list into a v-list and appending this vertical list to the main v-list; this is discussed in the section 'Switching from Horizontal to Vertical Mode').

We will focus our attention on the moment of switching from v-mode or iv-mode to h-mode and back again.

For the sake of simplicity we confine ourselves to the case where display math is not used inside a paragraph.

Switching from Vertical to Horizontal Mode

In this section we describe *when* and *how* level 3 of TEX accomplishes the change of modes from v-mode or iv-mode to h-mode.

First we say "when", i.e., we list the commands that — if executed in one of v-modes — switch TEX's state to h-mode.

Then we say "how", that is, we list the actions that TEX performs during the mode change.

Switching from vertical to horizontal mode: when. Some commands will be called here *vh-switches*, because if encountered in v-mode or in iv-mode they switch TEX to h-mode. They can be classified into two groups:

(a) explicit vh-switches:
  - *`\indent`;
  - *`\noindent`;

(b) implicit vh-switches (called by Knuth horizontal commands):
  - letter: any character token of category 11 (also implicit; for example, control sequence `\d` after executing the assignment '`\let\d=A`'; the assignment associates the token $\boxed{\texttt{d}}$ with a meaning that is primitive in TEX);
  - other character: any character token of category 12 (also implicit; for example, control sequence `\one` after executing the assignment '`\let\one=1`');
  - *`\char`;
  - a "chardef" token, i.e., a control sequence or an active character which has been assigned a meaning by the command `\chardef` (for example, control sequence `\ae` after the assignment '`\chardef\ae="1A`'; once again, the assignment associates the token $\boxed{\texttt{ae}}$ with a meaning that is primitive in TEX);
  - *`\noboundary` (a new primitive that appeared in TEX 3.0);
  - *`\unhbox`, *`\unhcopy` (independently of the contents of the box being an argument);
  - *`\valign`;
  - *`\vrule`;
  - *`\hskip`;
  - *`\hfil`, *`\hfill`, *`\hss`, *`\hfilneg` (these tokens are primitive, not macros, even though the effects they cause could be achieved using *`\hskip` with appropriate parameters);
  - *`\accent`;
  - *`\discretionary`, *`\-`;
  - *`\␣` (control space *`\␣` is a primitive command and if used in v-mode switches the mode to horizontal; note that normal space ␣, in general any space token, is ignored in v-mode);
  - `$` (also the first `$` of the pair `$$` starting the displayed math formula).

It should be stressed that commands *`\hbox`, *`\vbox` and *`\vtop` are not switches. Such commands encountered in v-mode do not change the mode. The box (preceded by proper glue) is appended to the current v-list.

Switching from vertical to horizontal mode: how. Assume that TEX is in either v-mode or iv-mode. When level 3 encounters a vh-switch at the beginning of the input token list it performs in turn the following actions:

(a) Optionally, a vertical glue `\parskip` is appended to the vertical list:
  - if TEX is in iv-mode and the list is empty, the glue is not appended,

- — if TeX is in iv-mode and the list is not empty, the glue is appended,
- — if TeX is in v-mode the glue is always appended to the part called "recent contributions" of the main v-list.

(b) If TeX is in v-mode (not iv-mode) the page builder is exercised, that is TeX runs the algorithm that moves elements of the v-list from the part of "recent contributions" to the part "current page". In particular it may cause page breaking (running the `\output` routine).

(c) Switching from v-mode or iv-mode to h-mode occurs.

(d) Variables `\spacefactor` and `\prevgraf` are assigned values 1000 and 0, respectively (these assignments are called by Knuth "global intimate assignments" and work in a rather peculiar way).

(e) A new h-list is initialised in the following way:
- — if the vh-switch that caused the mode change was *`\noindent`, the newly created h-list is empty;
- — if the vh-switch that caused the mode change was anything else (*`\indent` or any horizontal command), an empty box of width *`\parindent` is put at the beginning of the h-list.

(f) The following elements are appended to the beginning of the input token list:
- — the contents of the token register `\everypar` (normally this register is empty),
- — the vh-switch, provided it is a horizontal command; thus the explicit vh-switches *`\indent` and *`\noindent` are *not* put back into the input token list.

The rest of the input token list remains unchanged.

(g) Execution of the commands from the input token list starts. The commands are supplied by level 2 of TeX.

Answer to the Question Q1

Let us recall the question **Q1** of the first section. We have asked about the difference between

```
(*) \everypar{\def\indent{1}}
      \indent 3 is a prime number.
```
and
```
(**) \everypar{\def\vrule{1}}
      \vrule 3 is a prime number.
```

From the point (f) of the list of actions performed by TeX at the beginning of a paragraph (see subsection 'Switching from vertical to horizontal mode: how') we can draw the following conclusions: if a paragraph has started from the `\indent` command, the token $\boxed{\texttt{indent}}$ is not put back into the input token list, therefore after executing the actions (a)-(f) the input token lists differ in both cases.

In the case (*) the list is: '$\boxed{\texttt{def}}$ $\boxed{\texttt{indent}}$`{1}3␣is␣a␣prime␣number.`'; in the case (**) the list contains one more token: '$\boxed{\texttt{def}}$ $\boxed{\texttt{vrule}}$`{1}`$\boxed{\texttt{vrule}}$ `3␣is␣a␣prime␣number.`'.

Since redefining $\boxed{\texttt{indent}}$ has nothing to do with the remainder of the list, the typesetting result in the case (*) will be "3 is a prime number."

In the case (**) the token $\boxed{\texttt{vrule}}$ is first defined as a macro expanding to the token 1 and then the newly defined macro $\boxed{\texttt{vrule}}$ is expanded to 1. Therefore in this case the result will be "13 is a prime number."

This example shows some of consequences of the rule that the explicit vh-switches ($\boxed{\texttt{indent}}$ and $\boxed{\texttt{noindent}}$) are not put back into the input token list after switching to h-mode.

Switching from Horizontal to Vertical Mode

When level 3 of TeX executes commands in h-mode, some commands cause closing the h-list and performing some actions that lead to switching from h-mode to v-mode.

In the following subsection we say *when* TeX switches from h-mode to v-mode, i.e., we list the commands that cause switching. Then we explain *how* this mode change is performed.

Switching from horizontal to vertical mode: when. The commands listed below are called hv-switches, because if executed in h-mode they usually cause TeX to complete the h-mode and switch back to the enclosing v-mode or iv-mode. Similarly to the case of vh-switches, there are two groups of switches:

(a) explicit hv-switches:

— *`\par` (any token the current meaning of which is the same as the meaning of the token $\boxed{\texttt{par}}$ when TeX starts a job);

(b) implicit hv-switches (called by Knuth vertical commands):
- — *`\unvbox`;
- — *`\unvcopy`;
- — *`\halign`;
- — *`\hrule`;
- — *`\vskip`;
- — *`\vfil`;
- — *`\vfill`;

— *\vss;
— *\vfilneg;
— *\end;
— *\dump.

Switching from horizontal to vertical mode: how. The behaviour of TeX when it reads a hv-switch heavily depends on the type of the switch. If the switch is a vertical command (implicit hv-switch), TeX proceeds as follows:

— it inserts a token $\boxed{\texttt{par}}$ at the beginning of the input token list (*before* the hv-switch token), regardless of the meaning of the $\boxed{\texttt{par}}$ token;
— it starts executing commands from the input list (possibly expanding $\boxed{\texttt{par}}$ if currently it is a macro).

It should be emphasized that TeX *does not change* the mode before reading the token $\boxed{\texttt{par}}$ and that the expanded meaning of $\boxed{\texttt{par}}$ may redefine the token that triggered the action (please note the danger of looping).

If the switch is explicit (*\par), TeX "truly" finishes the paragraph, performing all or some of the actions (a) - (h) listed below.

TeX's behaviour depends on whether the h-list is empty or not at the moment. If the h-list *contains at least one element*, all of the actions (a) - (h) are performed. If the h-list *is empty*, only the actions marked with an asterisk are executed, i. e., (e), (g) and (h).

All possible actions are:

(a) discarding the final element of the h-list, provided it is glue or leaders;
(b) appending to the end of the h-list the following three elements:
  — \penalty10000 (forbid break),
  — glue of the size \parfillskip,
  — \penalty-10000 (force break);
(c) fixing the line-breaking parameters to be used in the next step,
(d) breaking h-list into lines and transforming this list into a v-list being the sequence of boxes, glue, penalty items and possibly other elements;
*(e) switching from h-mode back to the enclosing v-mode or iv-mode;
(f) appending the v-list created in step (d) to the enclosing v-list;
*(g) restoring the basic values of the parameters:
  — \parshape=0, \hangindent=0pt, \hangafter=1 (influencing the shape of a paragraph),
  — \looseness=0 (influencing the number of lines of a paragraph);
*(h) exercising the page builder if the current mode is the v-mode (but not iv-mode), i. e., initiating the process of moving elements from the recent contribution part of the vertical list to the current page.

Answer to the Question Q2

The question was:

What is the difference between:
 (∗) `\parindent=0mm \indent\par`
and
(∗∗) `\noindent\par`

Recall that we start in v-mode. The assignment of (∗) '`\parindent=0mm`' is just an assignment and does not append anything to the v-list. In both cases the command switching to h-mode (`\indent` or `\noindent`) causes appending the vertical glue of the size `\parskip` to the vertical list.

The command `\par` works differently in both cases (see subsection 'Switching from horizontal to vertical mode: how') because h-lists constructed are different:

 (∗) h-list at the moment of executing of the `\par` command contains a box of width 0 mm,
(∗∗) h-list at the moment of executing of the `\par` command is empty (the `\noindent` command does not append anything to the h-list).

So, according to what has been said in subsection 'Switching from horizontal to vertical mode: how', points (a) and (b), in the case (∗) TeX 'breaks into lines' a list containing:

— the empty box,
— `\penalty10000`,
— `\parfilskip` glue,
— `\penalty-10000`.

The result is a one-line paragraph that is appended to the v-list as a single box preceded by a `\parskip` glue and an interline glue.

In the case (∗∗) only the `\parskip` glue is appended to the vertical list, since the h-list is empty at the time the `\par` command is executed.

Answer to the Question Q3

We have asked what was the state of TeX after (∗) executing `\par` and after (∗∗) executing `{\par}`.

As we already know, TeX reacts to the command *`\par` performing the sequence of actions listed in subsection 'Switching from horizontal to vertical mode: how'. The results of most of the actions do not depend on the current level of grouping. However, the assignments mentioned in (g) are local within the current group.

Normally, at the end of each paragraph, TeX sets the values of `\parshape`, `\hangindent`, `\hangafter` and `\looseness` to 0, 0 pt, 1 and 0 respectively. But if a paragraph ends with `{\par}` instead of `\par` these values are assigned locally within the group surrounding `\par`. After closing the group TeX restores the values that the parameters had before the group started.

So, if the parameters mentioned above had standard values before `\par` or `{\par}`, their values do not change in both cases. If at least one of these parameters had a nonstandard value before `\par` or `{\par}`, executing just the `\par` command would result in restoring the standard value of this parameter, while in the case of `{\par}` the value of this parameter would be the same as before.

For example, by redefining `\par` as `{\endgraf}` and separating paragraphs with blank lines one can conveniently retain the same `\parshape` for several consecutive paragraphs.

Conclusions

We would like to emphasize that it is not the questions and answers mentioned in this paper that are important.

Our goal was to convince the reader that having a detailed (or, even better, formal) specification of TeX's mechanisms one could easily deduce the behaviour of TeX in all situations.

We have described here a small fragment of TeX's machinery. Although the description is only partial and not fully precise, we believe that it makes a lot of mysterious reactions of TeX understandable and straightforward.

Acknowledgements

Tomek Przechlewski and Piotr Pianowski: thank you.

NTS: The Future of TEX?

Philip Taylor
The Computer Centre, Royal Holloway and Bedford New College,
University of London, Egham Hill, Egham, Surrey, United Kingdom.
Internet: `P.Taylor@Vax.Rhbnc.Ac.Uk`

Abstract

Opinions on "the future of TEX" cover the entire spectrum, ranging from the definitive statement by Knuth — "My work on developing TEX ... has come to an end" — to proposals that TEX be completely re-written. In this paper, an intermediate position is taken, based on the fundamental premise that any successor to TEX must be 100% backward-compatible, both in terms of its behaviour when presented with a non-extended TEX document, and in terms of its implementation through the medium of WEB. A mechanism is proposed whereby extensions to TEX may be selectively enabled, and a further mechanism proposed which would enable conforming documents to determine which extensions, if any, are supported by a particular implementation. Finally, a proposal is made for an initial extension to TEX which would have implementation-specific dependencies, and mechanisms are discussed whereby access to such extensions could take place in a controlled manner through the use of implementation-independent and implementation-specific components of a macro library.

Introduction

Discussions on "The Future of TEX", both published and via the medium of e-mail/news-based lists, shew an enormous diversity of opinion: some would argue that Knuth's definitive statement that (paraphrased) "TEX is complete" leaves nothing further to be said, whilst others have advocated that TEX be entirely re-written, either *as* a procedural language or *in* a list-based language; in an earlier paper, I have myself suggested that one possible future derivative of TEX might be entirely window-based, allowing both input and output in textual and graphical formats. But events have occurred within the last eighteen months which have considerably influenced my point of view, and in this paper I present a far more modest proposal: that an extended TEX-based system (hereinafter referred to as *extended-TEX*, or *e-TEX* for short) be developed in a strictly controlled way, retaining not only the present look-and-feel of TEX but *guaranteeing* 100% backward compatibility with whatever truncation of the decimal expansion of π represents the most recent canonical version of TEX.

The reason for this change of heart dates from the 1992 AGM of DANTE (the German-speaking TEX Users' Group), to which I had the honour to be invited. There, Joachim Lammarsch, President of DANTE, announced the formation of a working group to investigate future developments based on TEX: the group was to be called the NTS group (for 'New Typesetting System'), to avoid any suggestion that it was TEX itself whose future was being considered, such activity being the sole remit of TEX's author and creator, Professor Donald E. Knuth. The group was to be chaired by Dr Rainer Schöpf, and included representatives of both DANTE and UK-TUG; Joachim emphasised that the group, although created under the ægis of DANTE, was to be a truly international body. An electronic mailing list, NTS-L, was announced, and participation was invited from any- and everyone throughout the world who wished to contribute to the discussion.

NTS-L proved a mixed success: it certainly attracted considerable interest, and in the early days discussion was almost non-stop; but it proved extraordinarily difficult to *focus* the discussion, and (like so many e-mail lists) the discussions frequently went off at a tangent... But then, after the initial burst of enthusiasm, discussions started to tail off; and as the time of the 1993 DANTE AGM came near, the only questions being asked on the list were "Is NTS dead?".

At about the same time, I was approached by Rainer, acting on behalf of Joachim who was indisposed, to ask if I would be interested in chairing the NTS group; Rainer felt (quite reasonably) that he had more than enough on his plate with his central involvement in the LaTeX-3 project (not to mention his real, paid, work!), and that he simply hadn't the time available to make NTS the success which it deserved to be. Needless to say, I viewed this offer with mixed feelings: it was a very great honour to be asked to chair such a group, but at the same time the group had already been in existence for nearly a year, and had apparently achieved nothing (in fact, it had never even met); would I be able not only to breath life back into the by now moribund project, but also go further and actually oversee the production of a realisation of NTS?

The more I thought about the problems, the more I became convinced that the key to success lay through simplicity: if NTS was ever to be more than a pipe-dream, a wish-fulfillment fantasy for frustrated TeXxies, then it had to be achievable with finite resources and in finite time; and if the results were to be acceptable to the vast number of TeX users throughout the world (a number which has been estimated to be at least 100 000), then it had to be completely backwards-compatible with TeX. Once I was convinced that I knew what *had* to be achieved, I also began to believe that it might be possible to accomplish it. And so, with some trepidation, I indicated to Joachim and Rainer that I would be honoured to accept their trust and confidence; I would agree to take over the NTS project.

But the road to damnation is paved with good intentions; and no sooner had I returned from the 1993 DANTE AGM, having once again had the honour to be invited to participate, than the spectre of TUG'93 began to loom large on the horizon; and the more work I put into its organisation, the more work it seemed to take. I was not alone — I willingly acknowledge the incredible amount of hard work put in by the entire TUG'93 committee, and in particular by Sebastian Rahtz — but the organisation of a multi-national conference, scheduled to take place at a University some 130 miles from one's own, is a mammoth undertaking, and one that leaves little time for anything, apart from one's normal, regular, duties. And, in particular, it left almost no time for the NTS project, to my considerable mortification and regret. But, by the time this paper appears in print, TUG'93 will be a reality, and, I hope, life will have sufficiently returned to normal that I will be able to devote the amount of time to NTS that the project so richly deserves.

But enough of the background: what matters today, and to this conference, is not how I as an individual partition my time; but rather what specific proposals I have for "The Future of TeX". I propose to discuss these under three main headings: *compatibility*, *extensions*, and *specifics*; under *compatibility* will be discussed compatibility both at the source (WEB) level and at the user (TeX) level; under *extensions* will be discussed a possible mechanism whereby extensions can be selectively enabled under user control, and a mechanism whereby an *e-TeX* conformant program can interrogate its environment in order to determine which extensions, if any, have been enabled; and under *specifics* will be discussed one possible extension to TeX which has been widely discussed and which will, in my opinion, provide the key to many other apparent extensions whilst in practice requiring only the minimum of additional *e-TeX* primitives. I must emphasise at this point that what follows are purely *personal* suggestions: they do not purport to reflect NTS policy or philosophy, and must be subjected to the same rigorous evaluation as any other formal proposal(s) for the NTS project.

Compatibility

What is compatibility? Ask a TeX user, and he or she will reply something like "unvarying behaviour: given a TeX document which I wrote in 1986, a compatible system will be one that continues to process that document, *without change*, and to produce results *identical* to those which I got in 1986". Ask a TeX implementor, on the other hand, and he or she will reply "transparency at the WEAVE and TANGLE levels; if *e-TeX* is truly compatible with TeX, then I should be able to use *exactly* the same changefile as I use with canonical TeX, and get a working, reliable, *e-TeX* as a result". Two overlapping sets of people; two totally different answers. And yet, if *e-TeX* is to be generally acceptable, and even more important, generally accepted, we have to satisfy both sets: the users, because without them the project will be still-born, and the implementors, because without them, parturition won't even occur! How, then, can we satisfy both sets? The answer, I believe, lies in the question itself: *e-TeX* must *be* TeX; it must use, as its primary source, the latest version of TEX.WEB, and it must make changes to TEX.WEB in a strictly controlled way, through the standard medium of a changefile; that is, *e-TeX* must be representable as a series of finite changes to standard TEX.WEB.

But if *e-TeX* is to be a changefile, how is the implementor to apply his or her own changefile as well? Fortunately there are several ways of accomplishing this: the KNIT system, developed by

Wolfgang Appelt and Karin Horn; the TIE system, developed by Dr Klaus Guntermann and Wolfgang Rülling; and the PATCH-WEB system, developed by Peter Breitenlohner. Each of these will, in varying ways, allow two or more change files to be applied to a single WEB source; thus the (system independent) changes which convert TeX.Web into e-TeX.Web can be implemented in one changefile, and the (system dependent) changes which implement *e-TEX* for a particular combination of hardware and operating system can be kept quite separate.

But this does not quite accomplish our original aim: to allow the implementor to use *exactly* the same change file for *e-TEX* as for TEX; in order to accomplish this, the changes effected by e-TeX.ch must be orthogonal to (i.e., independent of) the changes effected by <implementation>.ch; without a knowledge of the exact changes effected by each implementor's version of <implementation>.ch, such orthogonality cannot be guaranteed. None the less, provided that the changes effected by e-TeX.ch affect only the system-independent parts of TeX.Web, such orthogonality is probable, if not guaranteed; unfortunately, as we shall see, some proposals for *e-TEX* are guaranteed to conflict with this requirement.

So much for compatibility as far as implementors are concerned: what about compatibility from the point of view of the user? Here, at least, we are on safer ground: the users' requirements for compatibility are (let us remind ourselves) "unvarying behaviour: given a TEX document which was written in (say) 1986, a compatible system will be one that continues to process that document, *without change*, and to produce results *identical* to those which were achieved in (say) 1986". Thus (and here I intentionally stress an entire sentence) *the default behaviour of e-TEX must be* ***identical*** *to that of TEX, given a TEX-compatible document to process.* What does this imply, for *e-TEX*? I suggest two things:

- Every primitive defined by TEX shall have exactly the same syntax and semantics in *e-TEX*, and
- There shall be no new primitives (because existing TEX programs may depend on \ifx \foo \undefined yielding -true- for all currently undefined TEX primitives).

(gurus will appreciate that this is a considerable simplification of the truth, but I hope they will allow me this in the interests of clarity; clearly other constraints must obtain as well, for example identical semantics for category codes, and no additions/deletions to the list of context-dependent keywords).

Extensions

But given this as a definition of *e-TEX*, have we not backed ourselves into a black hole, from which there is no escape? How, if there are no new primitives, and all existing primitives are to retain their identical syntax/semantics, are we to access any of the *e-TEX*-specific extensions? I propose that we implement one, and only one, change between the behaviour of TEX and the behaviour of *e-TEX*: *if, on the command-line which invokes e-TeX, two consecutive ampersands occur, then the string following the second ampersand shall be interpreted as an* ***extension (file) specification***, in a manner directly analogous to TEX's treatment of a single ampersand at such a point, which is defined to introduce a ***format (file) specification***. Thus there is one infinitesimally small difference between the behaviour of *e-TEX* and TEX: if TEX were to be invoked as "TeX &&foo myfile", it would attempt to load a format called &foo; *e-TEX*, on the other hand, would attempt to load an *extensions-file* called foo — I suggest that the chances of this causing a genuine conflict are vanishingly small.

OK, so we have a possible way out of the black hole: we have a means of specifying an *extensions-file*, but what should go therein, and with what semantics? This is, I suggest, a valid area for further research, but I would propose the following as a possible starting point:

- if &&<anything> appears on the command line, then *e-TEX* shall enable one additional primitive, \enable;
- *extensions-file* shall commence with a record of the form \enable {options-list};
- *options-list* shall consist of a series of (?comma-delimited?) primitives and brace-delimited token-lists;
- if a given primitive occurs in the options-list to \enable, and if a meaning to that primitive is given by (or modified by) *e-TEX*, then henceforth that primitive shall have its *e-TEX*-defined meaning; (and if no such meaning exists, a non-fatal error shall occur);
- if a given token-list occurs in the options-list to \enable, and if that token-list has an intrinsic meaning to *e-TEX*, then the effect of that meaning shall be carried out; (by which we allow modifications to the semantics of *e-TEX* without requiring the creation of new, or the modification of existing, primitives; thus {re-consider partial paragraphs}, for example, might change *e-TEX*'s behaviour at top-of-page w.r.t. the partial paragraph which remains after

page-breaking; no new primitive is involved, nor are the semantics of any existing primitive changed). If the token-list has no intrinsic meaning to *e-TEX*, a non-fatal error shall occur.

Thus, by modifying only that area of the initialisation code which inspects the command line for a format-specifier, we allow for arbitrary extensions to the syntax and semantics of *e-TEX*. What we next need is a mechanism whereby *e-TEX*-conformant (as opposed to TEX-conformant) programs can determine which extensions, if any, have been enabled; thus a document could ascertain whether it is running in a TEX environment or an *e-TEX* environment, and modify its behaviour to take advantage of facilities which are available in the latter but not in the former.

In order for a program to be able to carry out this check in a manner which will be both TEX and *e-TEX* compatible, it must use TEX-compatible methods to check whether further *e-TEX*-compatible compatibility checks are supported: if we assume that the proposals above are implemented, then there is one reliable way of determining whether we are running (1) under TEX, or under *e-TEX* with no extensions enabled, or (2) under *e-TEX* with (some, as yet undetermined) extensions enabled:

```
\ifx \enable \undefined
     ... pure TeX, or e-TeX with
         no extensions
\else
     ... extended e-TeX
\fi
```

This relies, as does much existing TEX code, on `\undefined` being undefined; perhaps one extension implemented by *e-TEX* might be to render `\undefined` undefinable, just to ensure the integrity of such checks!

Once we are sure we are running under *e-TEX* with extensions enabled, we are in a position to make further environmental enquiries; but to do so will require an *a priori* knowledge of whether the environmental enquiries extensions have been enabled: a chicken-and-egg situation! Thus we need to proceed in a slightly convoluted manner, in order to ensure that we don't trip over our own bootstraps. Let us posit that, in order to enable environmental enquiries, we use something like the following in our extensions-file:

```
\enable {{environmental-enquiries}}
```

Then, in our *e-TEX*-compatible source (having ensured that we are running under *e-TEX* with extensions enabled), we need to be able to write something like:

```
\ifenabled {{environmental-enquiries}}
```

But we can't do this without first checking that `\ifenabled` is defined... Clearly this is becoming very messy (rather like one's first attempt at writing handshaking code for networking; how many times do you have to exchange *are-you-there/yes-i'm-here; are-you-there*s before it's safe to proceed with real data?). Fortunately, in this case at least, the algorithm converges after one further iteration: our TEX-compatible/*e-TEX*-compatible/totally-safe-environment-checking code becomes:

```
\ifx \enable \undefined
     ... pure TeX, or e-TeX with
         no extensions
\else
     \ifx \ifenabled \undefined
          e-TeX without the benefit
          of environmental enquiries
     \else
          ... e-TeX with environmental
              enquiry support
     \fi
\fi
```

(A similar approach could be used if environmental enquiries were implemented through the medium of `\enable {\ifenabled}` rather than `\enable {{environmental-enquiries}}`; it is a philosophical question as to which is the 'cleaner' approach).

One interesting issue, raised by the anonymous reviewer, remains to be resolved: if an *e-TEX* user decides to (a) enable some specific extension(s), whilst leaving others disabled, and (b) to dump a format file, what happens if that format file is loaded with a different set of extensions enabled? I have to confess that the answer to that question is unclear to me, and that an initial investigation suggests that extensions should *only* be permitted during the creation of the format file, not during its use; but that could have implications in the `\disable` functionality elsewhere referred to, and for the moment at least I prefer to leave this as a valid area for further research. Perhaps the whole extension/format area requires unification, and the enabling/disabling of extensions should simply become a part of the rôle of *Ini-e-TEX*.

Specifics

So far, I have concentrated on a generic approach to the question of *e-TEX*, and quite intentionally proposed only an absolute minimum of differences between TEX and *e-TEX*; but once the framework is in place, we are in a position to consider what features are genuinely lacking in TEX. This is a *very* contentious area, and one in which it necessary to tread warily, very contentious area, and one in which it is necessary to tread warily, particularly in view of Professor Knuth's willingness to regard TEX as complete: after all, if the creator and author of TEX sees no need for further enhancements, who are we, as mere users, to question his decision? Fortunately there is both precedent and guidelines; at the end of `TeX.Bug`, one finds the following text:

> "My last will and testament for TEX is that no further changes be made under any circumstances. Improved systems should not be called simply 'TEX'; that name, unqualified, should refer only to the program for which I have taken personal responsibility. — Don Knuth
>
> **Possibly nice ideas that will not be implemented.**
>
> - classes of marks analogous to classes of insertions;
> - `\showcontext` to show the current location without stopping for error;
> - `\show` commands to be less like errors;
> - `\everyeof` to insert tokens before an `\input` file ends (strange example: `\everyeof {\noexpand}` will allow things like `\xdef \a {\input foo}`!)
> - generalize `\leftskip` and `\rightskip` to token lists (problems with displayed math then);
> - generalize `\widowline` and `\clubline` to go further into a paragraph;
> - `\lastbox` to remove and box a charnode if one is there;
> - `\posttolerance` for third pass of line breaking.
>
> **Bad ideas that will not be implemented.**
>
> - several people want to be able to remove arbitrary elements of lists, but that must never be done because some of those elements (e.g., kerns for accents) depend on floating point arithmetic;
> - if anybody wants letter spacing desperately they should put it in their own private version (e.g., generalize the hpack routine) and NOT call it TEX."

Thus we have clear evidence that there are some possible extensions to TEX which Professor Knuth does not completely deprecate; he may not wish them to be incorporated in TEX, but I think we may safely assume that he would have no violent objection to their being considered for *e-TEX*.

But there is another source of information, too, in which he makes it plain that there is an area of TEX in which an extension would be deemed legitimate, and here (very surprisingly, in my opinion), he has suggested that the semantics of an existing TEX primitive could legitimately be modified *as part of the system-dependent changes to TEX itself*, without violating his rules for the (non-)modification of TEX. This arose during discussions between himself and others including (I believe) Karl Berry and Frank Mittelbach concerning the implementation of an interface to the operating system; Don suggested that it would be legitimate to extend the semantics of `\write` such that if the stream number were out of range (perhaps a specific instance of 'out-of-range', for safety, e.g., `\write 18 {...}`), then the parameter to that `\write` could be passed to the operating system for interpretation, and the results made available to TEX in a manner still to be defined.

When I first learned of this, I was horrified (and I still am...); not only is this a proposal to abuse `\write` for a purpose for which it was never intended (and in a manner which could wreak havoc on any program extant which uses `\write 18 {...}` to send a message to the console, which it is perfectly entitled to do (cf. *The TEXbook*, pp. 226 & 280)), it is a proposal to extend `\write` in a system-dependent manner. I found (and find) it hard to believe that Don could have acceded to these suggestions.

But these proposals received a wide airing, and were met by quite a degree of enthusiasm; not because people wanted to abuse `\write`, but because they were desperate for an interface to the operating system. Such an interface grants TEX incredible flexibility: one can sort indices, check for `tfm` files, in fact do anything of which the host operating system is capable, all from within TEX, and in such a way that the results of the operation become available to TEX, either for further calculation or for typesetting. Of course, there were also (very sound) arguments against: "what if the program performs a `$ delete [*...]*.*;* /nolog /noconfirm`?" was asked over and over again. (The command deletes all files to which the user has delete access, regardless of directory or owner, and recurses over the whole file system under VAX/VMS; there are equally powerful and unpleasant commands for most other operating systems.) What indeed? But if this feature were implemented through the abuse of `\write`, there would not necessarily be any provision for disabling it; and users would become legitimately paranoid, scanning each and every imported TEX document for the slightest trace of a system call, in the fear that computer viruses had migrated into their (previously safe) world of TEX.

Of course, TEX has never been truly safe; perhaps we are fortunate that the challenge of writing computer viruses appears not to be of any interest to those who are also capable of writing TEX (or perhaps those who have the intelligence to prefer, and to write in, TEX, have by definition the intelligence to see that writing viruses is a distinctly anti-social activity, and to refrain therefrom). I will not elaborate on this point, just in case it falls into the wrong hands...

And so, I propose that one of the first extensions to *e-TEX* which the NTS project should consider is the implementation, in a clean and controlled way, of a genuine `\system` primitive; implemented through the medium of `\enable`, it would be up to each individual user whether or not to allow of its use, sacrificing security for sophistication or preferring power and performance over paranoia. We might posit, too, a `\disable` primitive, so that even if the system manager had installed *e-TEX* with `\system` enabled, an individual user could choose to disable it once again (there are complications involved in this which I do not propose further to discuss here).

And once we have a `\system` primitive, we can then implement, through its medium, a whole raft of further extensions which have from time to time been requested by the TEX community (the following are taken almost verbatim from a submission by Mike Piff):

- Delete a file;
- Rename a file;
- Copy a file;
- Create a directory;
- Remove a directory;
- Change directory;
- Spawn a sub-process.

But these tasks are, by their very nature, incredibly operating-system specific; whilst I might type `$ delete foo.bar;`, another might write `%rm foo.bar` (I *hope* I have the latter syntax correct...); and surely one of the most important reasons for the use of TEX is its machine-independence: documents behave *identically* when typeset on my IBM PS/2 and on the College's VAX/VMS 6430. But if *e-TEX* users were to start hard-coding `\system {$ delete foo.bar;}` into their *e-TEX* files, machine-independence would fly out of the window; and *e-TEX* would have sown the seeds of its own destruction...

And so, I propose that for each *e-TEX* implementation, there shall exist a macro library which will be composed of two parts: a generic component, created by the NTS team, which implements in a system-independent manner each interaction with the operating system which is deemed 'appropriate' (whatever that means) for use by *e-TEX*; and a specific component, created by each implementor of *e-TEX*, which maps the generic command to the system-specific syntax and semantics of the `\system` primitive. The macro library is by definition easily extensible: if the *e-TEX* community decides that it needs a `\sys^delete_file` macro, and no such macro exists, it will be very straight-forward to implement: no re-compilation of *e-TEX* will be required.

Clearly there is an enormous amount of further work to be done: how, for example, is the `\system` primitive to return its status and results? What is to happen if `\system` spawns one or more asynchronous activities? Which of Don's "Possibly nice ideas" should be integrated into *e-TEX* at an early stage? How about the 'Skyline' question, or `\reconsiderparagraphs`? Should *e-TEX* be based, *ab initio*, on `TeX--XeT`? How are the NTS team to liaise with the TWG-MLC group, and with other interested parties? How are we to ensure that practising typographers, designers, and compositors are able to contribute their invaluable ideas and skills to the development of *e-TEX*? Some of these questions will, I hope, be debated openly and fully on NTS-L; others must be answered by the NTS team themselves (and here I have to confess that because of the pressures of this conference, the membership of that team is still in a state of flux). What matters most, at least to me, is that the philosophy and paradigms which characterise TEX are perpetuated and preserved for future generations: we have, in TEX, something very precious — the creation of a single person, Professor Knuth, which has had a profound effect on the professional lives of thousands, if not tens of thousands, of people; if we are to seek to extend that creation, then we must do so in a way which is entirely faithful to the ideals and intentions of its creator. I truly hope that we are up to that task.

A Future for TEX

Roger Hunter
TCI Software Research, Inc.
1190 Foster Road
Las Cruces, NM 88001
USA
Internet: `roger@nmsu.edu`

Abstract

The future of TEX is invisibility. The role of TEX should be similar to that of the microprocessor in a PC. The microprocessor is the heart of the system, but is completely invisible except for the sticker which says "intel inside." TEX must be made invisible with appropriate front-ends. These front-ends should emphasize the manipulation of content over appearance and reverse the trend toward WYSIWYG (What You See Is What You Get) interfaces with their emphasis on manipulation of appearance. Content-oriented interfaces provide far greater user productivity than WYSIWYG systems, and TEX is the ideal basis for such systems.

Introduction

TEX has a guaranteed future only if its use grows significantly. That growth can occur only if TEX is made much easier to use than it is now. Back-ends to TEX are necessary for any form of output so there are many of them. There must be strong pressure to create front-ends that make TEX much easier to use. The onslaught of WYSIWYG clickery makes the survival of TEX entirely dependent on good front-ends.

The Good and the Bad

Listing the good and bad features of TEX seems to be a favorite pastime of TEX lovers, and I am no different. The main difference in my list is that features often considered advantages are listed as disadvantages. First, the good features of TEX. The primary goals of Don Knuth's original TEX project head the list.

1. TEX produces superb output. This was Don's primary motive when he set out to create TEX.
2. TEX source is archival. The documents are in a standard ASCII form. The TEX language provides a linear, ASCII form which can be used as a standard for storage and interchange.
3. TEX is available on most platforms. This, together with its archival nature, ensures that TEX documents can be created and used anywhere there is a reasonably capable computer.
4. Many scientific journals accept compuscripts in TEX and provide style files.

And now the disadvantages.

1. The TEX language is a compromise. It has been said that the TEX language is understandable by everybody because instructions are written in plain English, not undocumented numerical codes. If Don Knuth had felt that the TEX language was the way we should read and write mathematics, then there would have been no need to create TEX, the program. Simply specifying the language would have been enough. The only justification for the form of the TEX language is as a linearized portable input to TEX, the program. In its present form, it is a compromise between the need to provide some support for direct entry and the need to process the result by computer. It would be wonderful to remove this compromise in favor of computer processing, but it is probably much too late.

   The two-dimensional mathematical notation evolved because it optimizes the use of the high-bandwidth human optical system. There are many mathematical expressions which are virtually impossible to grasp in TEX input form — all are instantly comprehensible in TEX output form.

   Publishers had hoped that TEX would be the solution to the rising cost of typesetting, and now they are not so sure. A major reason for this is that authors do not write style-independent TEX code. Leslie Lamport defines and discusses visual design and logical design in the LATEX *User's Guide & Reference Manual.*

Logical design is the key to good TEX documents, but without a way to enforce it, most authors end up with a large component of visual design in their code. This is a nightmare for publishers who need to typeset using a specific style, and is the main reason why author submissions are so costly. It invariably costs more to use the author's original TEX document than it does to re-key the entire thing. If the publishers will not champion TEX, the future cannot be bright. Stamp out abominations like `\it word \rm`!

TEX will have no future unless authors are completely isolated from the TEX input language. Although it is archival, the TEX language is unfit for humans. A proper front-end eliminates these problems.

2. TEX is in the public domain. As wonderful as this may seem at first, it means that now that its creator has stopped working on it, everybody wants a say. We have gone from a committee of one to a committee of the entire world. How much progress can a committee of this size make?
3. TEX is extensible. This is marvellous for developers of macro packages and styles. It is a disaster in the hands of authors. Authors delight in creating new sets of macros and in using them inconsistently in a document. Publishers find it much cheaper to re-key an entire document than to rewrite an author's macros to fit a style.

   The policy of the American Physical Society, the American Institute of Physics and the Optical Society of America on submitting documents is the right one. To have a paper accepted, you must use the REVTEX styles, but most importantly, you are prohibited absolutely from defining and using macros. Amusingly, the REVTEX guide carefully explains that there are two classes of macros, and then states flatly that you cannot use either kind!
4. TEX is stable and unchanging. Whatever the arguments or the reality, making this statement gives TEX a dead feel. Even if TEX itself does not change for the forseeable future, the continued development of packages like LATEX3.0 provide the necessary life. This is mostly a matter of public relations.

TEX Must be Invisible

TEX is the microprocessor, LATEX is the operating system, and appropriate front-ends are the application programs. Just as the average user has no need to know how a microprocessor works, and a user of an application program needs only a rudimentary knowledge of the operating system, the average user should never be exposed to TEX. These days, most drivers of cars do not know how an engine works. Although knowing how an engine works may somehow make you a better driver, requiring that you know how an engine works would be ridiculous. It would ensure that most people would not drive. We have roughly the same situation for TEX. Requiring that users know TEX will ensure its demise. A technology is mature when most of its users do not know how that technology works. We should strive to make TEXnology mature.

The High Cost of Visual Design

A recent study estimates that 2% of the United States gross domestic product is lost through unproductive use of computers. At the head of the list of offending behaviors is "font futzing" — endless fiddling with the appearance of a document. A section head at Sandia National Laboratories told me that his researchers spend huge amounts of time preparing reports using WYSIWYG Windows word processors. They spend most of the time changing fonts and page layout. When the documents are submitted, they must be reformatted to fit the required style. The process takes hours because all of the formatting is local and visual.

The same effects exist in the TEX world. Most of us are familiar with people who fall in love with TEX and run around saying, "Look at this incredible effect I just produced" or, "Look at this fantastic macro I created." Highly paid professionals endlessly playing with TEX macros to get just the right visual effect are wasting their time doing work that is unproductive and for which they are not trained. The only way to avoid this problem is to provide a front-end which enforces or strongly encourages the principles of logical design.

Interface is Everything

Given that invisibility of TEX is essential and that logical design has a large productivity payoff, interface is everything. Attractive interfaces are the reason WYSIWYG word processors are simply taking over. They are addictive. Their addictive nature and their total focus on visual design makes them one of the most insidious productivity sinks in existence today. The salvation of TEX lies entirely in the development of good interfaces, and those interfaces must encourage and, if necessary, enforce logical design over visual design.

The main reason for using TEX instead of one of the leading word processors is to obtain the far superior output. Because there are no alternatives, you are willing to put up with the input language. The situation with symbolic systems like Maple and Mathematica is the same. The benefits of these systems must outweigh the disadvantages of their unnatural user interfaces before someone will choose to use them. This restricts use to a tiny fraction of the potential audience. By making the interface much better, the number of TEX users could be increased several orders of magnitude. The same argument applies to Maple and Mathematica.

The Right Interface

The essential features of a good TEX interface are as follows:

- The interface must encourage authors to work directly at the computer.
- The interface must encourage logical design over visual design.

The penalty for using the computer over the blackboard or a pencil and paper should be minimal. The language you use to read and think about your document should be the language you use to enter it into the computer. The time taken entering TEX codes is wasted time which could be used for developing the content.

The current crop of Windows-based word processors (Word for Windows, Ami Pro, and Word Perfect are the three most popular) define WYSIWYG. The essential feature is visual design. One manifestation is that you are encouraged to interact with an image of the printed page. Another is that you select text and give commands which determine the appearance such as the font face, point size, and weight.

The fact that all of the best-selling word processors use a WYSIWYG interface has lead to the perception that there is no other way. In fact, the use of a GUI (Graphical User Interface) has become synonymous with WYSIWYG. The result is that millions of people are forced to view crude representations of the printed page through screen windows which never match the pages. At the same time, they have come to spend much of their time at the computer worrying about page layout and typography.

Interfaces which emphasize logical design provide a much better way to create, edit, and interact with documents. The main features of a logical interface are as follows.

- Lines are broken to the screen window.
- You select text and designate it as a section head or apply an emphasis.
- Fonts and colors used on the screen are chosen to maximize screen readability and are independent of the choices made for the printed output.

Just as there is a perception that GUI implies WYSIWYG, there is a corresponding perception that logical implies linear. People seem to think that an interface which uses logical design requires that you enter obscure codes to get the results you want. The primary example in the TEX world is the notion that using the TEX input language directly is the only right way. This is simply false — it is possible to create a logical interface which displays and has you interact with mathematics in its natural (TEX output) form.

Some Interface Issues

TEX is a batch system. There are a number of interesting problems which arise when you consider implementing a much more interactive system.

The first problem has to do with TEX's line breaking algorithm. I have often heard people say that the ultimate system would allow you to interact with pages in the way you do with a WYSIWYG word processor, but the page layout would be updated instantly using TEX. Even if you translate this desire to a logical system, there are drawbacks. For example, you could be typing or editing toward the end of a paragraph and have all of the lines above you in the paragraph jiggling about as you type. This is because TEX's line breaking algorithm can change the breaks throughout a paragraph when you make a change anywhere in the paragraph. The effect could be very distracting.

Another question which simply doesn't arise in a batch system has to do with spaces. Who owns the spaces? When TEX puts extra space around operations, relations and punctuation in batch mode, the question makes no sense. When you are dealing with an interactive system, the insert cursor must be placed somewhere, and the choices made have a significant effect on the feel of the system. For example, where should the cursor be placed as you move through the expression $x + y$? TEX inserts extra space around binary operations. Should the cursor position between x and $+$ be next to the x, next to $+$, or somewhere in between? If you take the position that the $+$ owns the extra space, then the cursor should be placed next to the x. This seems like a very minor point, but it has a large effect on the feel of the system.

Blue Sky's Lightning Textures provides a way to enter TEX codes and see the resulting TEX output almost instantly. This is a completely different approach which I view as complementary to the interface I have described. Lightning Textures provides the greatest value for typesetters and others performing high-end layout work. The interface I have described is meant for authors.

Scientific Word

Scientific Word is a Windows-based scientific word processor based on the principles that I have mentioned. It provides a logical interface to documents and stores LaTeX files. It includes Richard Kinch's TurboTeX for previewing and printing.

Experience with users of *Scientific Word* has been very interesting. Initially, many users feel extremely uncomfortable with the fact that they are not interacting with a page image. They spend a great deal of time previewing to see if they really will get the results they want. As they continue to use the system, the frequency of previews decreases. Once they have learned to trust the system, they relax and focus on the content exclusively. Only in the final stages do they concern themselves with the printed form. The habits developed by using WYSIWYG systems are difficult to break, but once they have been broken, users realize how much more productive they can be.

Direct interfaces between *Scientific Word* and symbolic systems are also being developed. An experimental version of *Scientific Word* lets you interact directly with Maple using the same principles employed for TeX. Maple's input language is invisible in this system — the notation for input and output is the standard, natural, mathematical notation.

LexiTeX: Context-Sensitive Legal Citations for LaTeX

Frank G. Bennett, Jr.*
Law Department
School of Oriental and African Studies
Thornhaugh Street
London WC1H 0XG
Internet: `fbennett@clus1.ulcc.ac.uk`

Abstract

The most widely used style guide for legal publications in the United States is *A Uniform System of Citation* (the so-called 'Blue Book'). LexiTeX is a LaTeX style which automates typeface selection, citation truncation, citation cross-referencing, and the production of tables of authorities in the Blue Book style. This article discusses the design problems encountered in coding LexiTeX, points out a few of the dirty tricks which were used to smarten up its response to context, and comments on further work that waits to be done in this area. A reference card for use with LexiTeX is included.

The Blue Book

Law reviews in the United States are a curious corner of the publishing world. Traditionally, most such journals are edited entirely by law students, who compete in their first year of study for admission to staff positions. Participation in law review, particularly in an editorial post, is viewed by potential employers as an important token of accomplishment in making hiring decisions. And so it goes that despite the long hours and lack of pay, the competition for these positions is quite intense.

The most widely used style in U.S. law reviews is *A Uniform System of Citation*[1] (the so-called 'Blue Book'),[2] compiled and periodically revised by staff at the Columbia Law Review, the Harvard Law Review Association, the University of Pennsylvania Law Review and the Yale Law Journal. The Blue Book requires that the origin of each proposition asserted or referred to by the author be precisely identified in a footnote citation. It lays down detailed and specific rules concerning typeface conventions, short-form citations and cross-referencing. All of these rules vary according to the nature of the source as well as the context in which the citation appears, which makes the proofreading of legal citations a particularly unenviable chore.

Indeed, in the eyes of some, the Blue Book's rules are specifically tailored to test the persistence and will power of those lucky enough to find their waking lives revolving around its creed. Nonetheless, there is enough underlying consistency in its conventions that typeface selection and the formatting of short-form citations can be almost completely automated. The key to context sensitivity as required by the Blue Book is the ability to remember key portions of the text on a start-to-finish reading. Both law review editors and TeX are capable of doing this; the difference is that TeX is a good deal faster, and rather less likely to develop indigestion.

The Reference Card at the end of this article, together with the tables illustrating the use of style blueprints and citation templates, should contain all of the information necessary for ordinary use of LexiTeX. The flow diagram in the Reference Card should help the user to visualize how the style operates. If I have accomplished my original purpose in writing the code of LexiTeX, it should be possible for a reader who is generally familiar with both LaTeX and the Blue Book to make a copy of the Reference Card and the tables, read no further than the end of this sentence, and immediately begin using LexiTeX productively in his or her own work. This should be

* The author wishes to thank Boris Pevzner and Steve Bellovin for their patience with and comments on the inital beta 1.0 release of LexiTeX and its friends. The (as yet unrealized) BibTeX analogue discussed under Further Work was suggested by Steve Bellovin. Gratitude is also due to George Greenwade for his encouraging words on the initial release, and to Andrew Hoddinott for introducing me to TeX in the first place.

[1] A UNIFORM SYSTEM OF CITATION (14th ed. 1991).

[2] With the emphasis on 'blue'. With the opposite stress, the term refers to a rather different type of literature.

the first and final test of LexiTeX's utility. The Blue Book is bad enough by itself; if make-easy software for it requires a lengthy manual, one might as well not bother.

On the slightly reckless assumption that I *have* accomplished my original purpose, the scope of this article is limited to a general overview of the design of LexiTeX, with a few ancient mariner's tales of problems that cropped up in the course of simultaneously drafting the style and learning the rudiments of the TeX language. The following section explains how LexiTeX attempts to reduce the citation formatting process to a manageable set of units. This is followed by a discussion of some of the more entertaining problems dealt with by the code. The article closes with some comments on further work that is waiting to be done in this line.

The LexiTeX Style Engine

The format of any Blue Book citation can be fully described in terms of six essential *text units*, and nine punctuation *bridges* (seven, actually, two of which have a plural 'alter ego', for a total of nine in all). The text units each require up to two associated typefaces, which I will refer to as the *main* and *alternative* typefaces for each text unit. The presentation form of any Blue Book citation can be described in terms of text units, associated typefaces, and bridges; and the various mutations of the citation in subsequent references can be accomplished by manipulating units and bridges, without reference to their actual contents. This is what makes the automation of citations possible.

In order to combine flexibility with ease of use in an automated system,[3] we can (1) define a macro which, from a blueprint of necessary details, (2) defines another macro which in turn, from a set of text arguments, (3) defines macros to stand for individual citations. In step (2), only the text need be fed to the macro; everything else can have been defined in advance. After once feeding the details of a particular source to a macro in this way, it should be possible to reduce subsequent references to it to a single, context-sensitive nickname.

Six essential text units

1. The author's name;
2. The title of the work;
3. The source in which the cited work is located, including any volume number;
4. The page on which the cited work is located in the source;
5. An optional reference to a particular page or section number of the work; and
6. The tail end of the citation (usually a year of publication, perhaps with an indication of the editor, translator, publisher and city of publication as well, all usually enclosed in parentheses).

Nine punctuation bridges

1. Author-to-Title
2. Title-to-Work
3. Work-to-Source
4. Source-to-Location-page (singular and plural)
5. Location-page-to-Specific-reference-page
6. before-the-Tail-end
7. after-*Id./supra* (singular and plural)

To state this in shorthand terms, the first, most general macro in this series can be thought of as a 'style template', and the macros defined by it as 'citation templates'. The citation templates produce 'nickname macros', which in turn are used to produce actual printed citations in the form appropriate to a given context.

The blueprints for use by the style template are all contained in the file `lexicite.tex`. Additional blueprints can be added to this file by the adventurous user as required. In normal circumstances, the user need only determine which citation template is appropriate to a given citation, and feed it the text of the citation as a set of arguments, along with a nickname that will stand for the citation in subsequent references.

Fun in the Sun

Once this basic structure is in place, it is largely a matter of fine-tuning to bring the output of the package into line with Blue Book requirements. This section discusses some of the tricks and kludges used to that end. It is not a comprehensive discussion,

[3] It should be mentioned that LexiTeX is not the first attempt to automate short-form citations. See the materials prepared by David Rhead in connection with the LaTeX3 project: Rhead, *Towards BibTeX style-files that implement principal standards*, TeX LINE 12 (May 1990); Rhead, *How might LaTeX3 deal with citations and reference-lists*, TeXLINE 13 (September 1991). The code behind this proposed interface, as I understand it, defines each style of citation separately. Through the use of blueprints and the style template macro `\@law@newcitestyle`, LexiTeX attempts to provide a more general, customizable interface.

Sample `lexicite.tex` blueprint entry for LexiTeX

```
\newcitestyle{newarticle[a]}%
{Typefaces:author(x)[b]:title(i)[b]:cite(s)[b]/Citetype(a)[c]}%
{Bridges:author(,\ )[d]title(,\ )[d]cite(\ /\ )[d]page(,\ )[d]refpage(\
)[d]afterid(\ at~/\ at~)[d]}%
{6}[e]%
{{#1}[f]{#2}[f]{#3}[f]{#4}[f]{#5}[f]{(#6)}[f]}%
```

[a] This gives the name of the citation template to be generated.

[b] Within these sets of parentheses, for the author, title and first-cite-part portions of the citation, `s`, `b` and `i` select small caps, boldface and italic type as the main typeface. `S`, `B` and `I` respectively select each of these as a typeface which may be specially selected using `\\...\\`. The parentheses may *not* be left empty. To use the default (roman) typeface, place some other character here.

[c] Within this set of parentheses, `a`, `b`, `c` and `s` will classify all citations generated by this citation template as being to articles, books, cases or statutes, respectively. These parentheses must *not* be left empty.

[d] These parentheses contain the citation bridges that will be placed between the portions of the citation indicated. Two bridges, singular and plural, are separated by a / in the parentheses following `cite` and `afterid`. The bridges following `afterid` are used to attach specific page references to *Id.* and *supra* citations. All of these parentheses may be left empty (provided that the plural separator / is not omitted from the two fields to which it applies).

[e] This argument states how many arguments the finished citation template will accept. It must be a number between 1 and 6.

[f] Six pairs of matched braces must appear inside this argument. Any other than the first (which represents the citation nickname) may be left empty, but the bridges must of course take this into account. The number of arguments inserted here must correspond with the number stated on the line above.

but may help those interested in making improvements or alterations. Those wishing to dig deeper on the technical side are invited — indeed, positively encouraged — to have a look at the code itself, which is available from the major TeX archives. Those who want a friendlier introduction to the care and feeding of LexiTeX are gently referred to the penultimate paragraph of the first section of this article — after which they are invited to suggest improvements.

Eliminating redundancy. When I began drafting the code of this style, the file `lexitex.sty` quickly grew to about 30 kilobytes, roughly the same size as in the current release. But in its early life, LexiTeX had separate storing and printing routines for each of the four general classes of citation; and as new features were added it became increasingly cumbersome to carry the changes through to each set of routines.

Eventually the pressure of common sense overwhelmed me, and these separate routines were collapsed into a single set of macros which can respond differently in minor ways depending upon a toggle indicating the citation type. While this is little more than good programming practice, it was surprising to see the extent to which a job that initially seemed to bristle with nasty real-world edges could be distilled into a compact and unified logical structure.

The code will no doubt tolerate a good deal more optimization and trimming.

Expansion control. While it is processing a job, LexiTeX swaps a great deal of information between macros. Much of this ends up being printed in a variety of LaTeX environments, or is exported to external files. Rigorous expansion control is therefore crucial if undesired smash-ups are to be prevented. And in some places `\noexpand` and `\the` statements were not enough by themselves to prevent gastric chaos.

To cope with such situations, LexiTeX provides the `\@law@clean` macro, which accepts a token register and an arbitrary control sequence as its arguments. When used, it first defines a number of common control sequences which expand into TeX primitives (which cause difficulties if expanded at certain crucial stages of LexiTeX's ruminations) as strings of themselves: `\string\LexiTeX`. The definition is stored under the arbitrary control sequence name *locally*, so that its scope can be limited by grouping. The control sequence can then safely be used to stand for the token register in LexiTeX's internal code. The `\@law@cleanup` macro runs `\@law@clean` over all token registers which might contain problematic control sequences. If special character or formatting commands within a LexiTeX argument give an error during processing, this may be due to the absence of a TeX primitive invoked by this command from the list of temporary definitions in `\@law@clean`. If adding a suitable entry covering the command to the list makes the problem go away, please send me the details, and I will include the patch in future releases.

Output transformations. The citation templates produce two macros for each citation, `\<blah>` and `\<blah>full`, for short- and full-form citations respectively. The `\<blah>full` macro is invoked automatically immediately after a citation is declared using the citation template, unless output is suppressed by the user. Afterwards, the short-form `\<blah>` will normally be used; the `\<blah>full` form can be quietly ignored, but it is there for what it is worth.

Short-form output in the text must be context-sensitive. For most governing conditions, this is pretty pedestrian: a variable or a macro marks the occurrence of an event, such as a footnote number or the name of the immediately preceding citation, and the short-form print routine is set up to adjust the elements of the citation accordingly. One Blue Book rule does present special difficulties, however.

In short-form references to articles and books for which an author is specified, the title or name of the work should be omitted (automatically, that is), *unless* some other work by the same author in the same typeface has been cited already. The problem is that LexiTeX's citation macros can only be expanded one at a time, and they are designed to do just one thing — make a citation *now*. An actual search for a matching author would be slow, and it would require significant and cumbersome re-design of portions of the code. Not fun.

In the end there was a much simpler solution. The author's name *itself* can be defined as a macro, by first running the register containing the author's name through `\@law@clean`, and then using `\csname\endcsname` to make a macro of it. This routine is made part of the citation template. The first time around, it is defined to expand to 1, but if it is found to be defined already, it is set to expand to 2. With this information, the print routine can honor the Blue Book rule without knowing which nickname citation is associated with the matching author name.

Looking ahead for signals. The one element of a citation that cannot be stored by the citation template is the reference to the specific page or section number where a particular proposition occurs in the cited work. This page number must be added after the nickname macro is created, and it must be incorporated into the citation before it is printed. The goal is to adopt a syntax where:

```
\bluebook+{21}.
```

will expand into the full citation form:

A Uniform System of Citation 21 (1991).

or into the appropriate short form:

A Uniform System of Citation, *supra* note 1 at 21.

This requires that the nickname macro look ahead in the document before it prints its contents. For the most part this is straightforward. After resetting the appropriate registers to reflect the fixed information contained in the citation, nickname macros look ahead with LaTeX's `\@ifnextchar` command to see if the character coming up satisfies any of a series of conditions. If the next character is * or -, the print routine is bypassed, and printing is suppressed.[4] If the character is +, LexiTeX gobbles the +, and checks further to see if there is a second +. If there is, it knows that the upcoming argument refers to multiple pages or sections. If there is not, LexiTeX checks for a -, which would indicate multiple references with the use of a singular bridge (as in Federal Constitution sch. 5, paras. 2 & 4 (1957)). If *that* is not present, then the reference must be to a single page or section.

From LaTeX's own command syntax, it was clear that this kind of conditional testing is possible in TeX, but before I could implement it in LexiTeX I needed to learn a fairly simple lesson about how TeX churns through a document. The difficulty I experienced is one that people who, as was my own case, come to TeX from higher-level procedural languages are likely to experience, so perhaps it is worth mentioning.

The LaTeX command `\@gobble` is a simple macro that accepts one argument and does nothing with it: `\def\@gobble#1{}`. It is necessary to ensure that `\@gobble`, when used, expands *immediately* before the character or argument that is to be dispensed with. But the TeX language does not have a means of distinguishing between the document 'itself' and the macros that expand within it; it's all so much program code. A complex set of macros such as LexiTeX therefore requires careful structuring to ensure that

[4] This feature might be required where a non-conforming citation is used for a cited work, such as citations which are part of a quotation.

a \@gobble statement is the very *last* operation performed in any chain of macros leading up to it. In particular, it cannot be nested within an \if...\fi condition, because it would chew up part of the condition itself, causing perhaps all sorts of strange and wonderful things to happen.

LaTeX's \@ifnextchar hack works by storing either the first or the second of the two alternative arguments shown to it in a particular macro, and expanding this stored internal macro as its last act. LEXITEX's \@law@argscheck routine uses \expandafter within the appropriate alternative argument to \@ifnextchar to run \@gobble as its 'last' act, before invoking the next command that needs to be chained. And so on back to the level of the document itself.[5]

Because the text of the initial full citation produced by the citation templates is actually written by the \<blah>full macro that they themselves define as their own last act, +{<page number>} can be added at the end of a citation template declaration as well as after a nickname reference.

This overall scheme reduces the typing required to specify page numbers to a minimum, and allows the same syntax to be used for this purpose for all types of citations. That in turn helps increase the speed with which a document can be prepared.

Producing tables. Once the details of a citation have been stored in a macro, it is a (relatively) simple matter to send it to an external table for processing by makeindx.

Every citation in LEXITEX is flagged as being to a book, an article, a case or a statute. You can use LEXITEX to produce lists of all citations falling into any of these four classes, simply by placing any of \makebooktable, \makearticletable, \makecasetable, or \makestatutetable before \begin{document}. This produces the files *.btb, *.atb, *.ctb and *.stb respectively.

Each of these files must then be sorted, using the makeindx utility. The makeindx program should be instructed to use the appropriate index style file for that type of table. These are, respectively, lexibook.ist, lexiarti.ist, lexicase.ist and lexistat.ist. The output of each sort should be directed, respectively, to *.bok, *.art, *.cas or *.sta.[6] Finally, the command or commands \printbooktable, \printarticletable, \printcasetable or \printstatutetable should be placed where the tables are to appear when LaTeX is run over the document a final time.

Apart from producing tables for actual publication, the automatic generation of tables allows citations of each class to be gathered into a single location for proofreading. This can save a great deal of time and eyestrain.

More expansion control. For reasons which I readily admit to understanding only hazily, LaTeX stubbornly refused to cooperate when I attempted to place macros or \the statements inside a file-write macro based on LaTeX's own index-writing routine. The difficulty was related to expansion; certain control sequences apparently change their meaning between the time the file-write macro is run and the time of the actual export via LaTeX's output routine.

It seemed as though I had a choice of either suppressing expansion altogether (in which case the information in the registers or macros was often replaced before the write statement actually went into operation at the end of the page — resulting in a string of citations to the last source on the text page), or using \immediate to avoid the strange collision with the output routine (in which case the current page number written to the table might not be correct).

The solution adopted in the end was to use the \@law@cleanup macro, discussed above under the subheading Expansion Control. \@law@cleanup is a bit of a kludge, but it has the advantage of allowing us to selectively control the expansion of macros unknown to LaTeX without letting them boil themselves down to incomprehensible primitives before they end up in the external file.

Sorting the citations. Some materials need to be sorted in special ways, which cannot be derived easily from the citation. For example, some decisions of foreign courts ought to be sorted first by the name of the court, then by the date of the decision. Edited books, too, should be sorted by the name of the editor — which, in the Blue Book style, occurs in the tail end portion of the citation.

[5] Those following the trail back upstream in the code will find that it goes cold at this point. This is because \@law@argscheck is inserted into the nickname macro by the citation template by first storing it to a temporary token register \@ltok@a and using \the to invoke it. This is necessary because it falls within the scope of an \xdef that is necessary in order to "freeze" the current registers and macros in the nickname's definition. The use of \the prevents \@law@argscheck from actually expanding during this definition.

[6] This has only been successfully tested with the DOS version of makeindx that is supplied with emTeX. The long lines generated by LEXITEX may cause problems for some versions of makeindx.

The easy solution adopted for this problem is to include the nickname of each citation in the output to the table file. The nickname is unique, and the citations therefore come out in the desired order, if the nicknames were chosen with this sorting process in mind. The macro of the form `\lexicite{<blah>}` can be used to invoke a citation nickname that contains a number, such as a date. When the sorted file is read by the relevant `\print*table` command, the nickname is ignored.

Selective appearances. The user may wish to selectively control the appearance of certain information in the citations and tables. For this purpose, LexiTeX uses `^` and `_` as active characters.[7] The syntax for these is given in the reference card, and the principle of their use is straightforward. In all of the five main citation elements other than the nickname, the form `^...^` may be used to enclose text that will appear in the full form of the citation, and be eliminated both in the tables and in subsequent short-form references. This is useful where the citation must include some explanatory note on its first occurrence. For example:

```
\newinbook{rajarole}{Y.A.M. Raja Azlan
Shah^ (as he then was, being later
elected {\emYang di-Pertuan
Agong\/})^}{The Role of Constitutional
Rulers in Malaysia}{\\chapter 5 of\\ The
Constitution of Malaysia:  Further
Perspectives and Developments}{}{F.A.
Trindade \& H.P. Lee, eds.  1986}[8]
```

In this example, the explanatory note of the author's personal title will appear only in the first occurrence of the citation.

In optional page and section number references appended using +, -, ++ and +-, both `^...^` and `_..._` can be used. In this context, any text within `^...^` will appear only in the text, and not in the tables, while `_..._` conversely suppresses enclosed material in the text, but does export it to the table. This is useful for limiting the level of detail in statutory references within a table of statutes. A complex example might run as follows:

```
\constsch+-{{5, para^s^.~5} \& {_5,
para.~_8}}[9]
```

If a table is produced from this citation, it will contain two entries:

```
\statutetableentry{{constsch}{{\rm
  \ltokspecialface ={\sc }Federal
  Constitution}{\rm \ltokspecialface =
  {\sc }}}!{\ sch.~5, para.~5}}{1006}
\statutetableentry{{constsch}{{\rm
  \ltokspecialface ={\sc }Federal
  Constitution}{\rm \ltokspecialface =
  {\sc }}}!{\ sch.~5, para.~8}}{1006}
```

Iterative file writes. Citations to statutes often refer to more than one section. References to multiple page numbers of other types of works do not cause any problem, because these specific page references are ignored when the entry is written to the external table. But for statutes, each section referred to becomes a sub-item in the statute table. It would not do to have a string of sections written as a single entry.

If the user enters the punctuation between section numbers as `\,`, `\&` or `\dash`, rather than simply `,`, `&` or `--`, each section reference will be separately written to the statute table file, as illustrated above. Note, however, that `makeindx` performs an alphabetic, not a numeric sort of the sections; you may need to shift the order of entries around by hand before the `*.sta` table is finally printed.

Further Work

In its current form, LexiTeX has realized most of the goals of its author. Citations can be defined with a wide variety of typeface combinations. Subsequent abbreviation is sensitive both to context and to the type of citation. Page and section numbers are simple to enter, and are correctly integrated into the final citation. Bibliographic tables can be produced automatically, and multiple statutory section numbers are easily entered and correctly parsed for entry into the table of statutes. Alas, there is always room for improvement.

Memory requirements. Because LexiTeX stores all the information required to produce each citation in TeX's memory as the document is processed, it has a way of running out of space on large documents. A large TeX solves the problem for any document of reasonable length, but the style could be made less memory-hungry. The most obvious area of redundancy is in the paired `\<blah>` and

[7] Please note that this may cause conflicts with other styles. The next release should offer the option of using ordinary control strings for this purpose, rather than active characters.

[8] This expands in the text to: Y.A.M. Raja Azlan Shah (as he then was, being later elected *Yang di-Pertuan Agong*), *The Role of Constitutional Rulers in Malaysia*, CHAPTER 5 OF THE CONSTITUTION OF MALAYSIA: FURTHER PERSPECTIVES AND DEVELOPMENTS (F.A. Trindade & H.P. Lee, eds. 1986)

[9] This expands in the text to: Federal Constitution sch. 5, paras. 5 & 8.

`\<blah>full` macro definitions. Most of the information stored in both of these macros could be hived off into the master storage macro for that citation name, say `\@<blah>`. Then this storage macro can be recalled by either of the citation nickname macros, which would then only need to add or alter any bits and pieces necessary to their own purposes.

The design of LexiTeX emerged as its author struggled to learn the basics of TeX programming; it is therefore a certainty that there is room for memory savings and optimization in other parts of the code as well.

Emacs Lisp support. The Lisp extension language for the Emacs editor is powerful enough to provide context-sensitive support for LexiTeX itself. That is, prompting for command completion and arguments could be made dependent upon the contents of the file `lexicite.tex` and the citation template declarations made to the current point in the document.

I am not a Lisp programmer myself, but the essentials of such a set of routines would be:

1. The ability to scan the `lexicite.tex` file and, for each citation style, identify:
   - The style name;
   - The number of arguments required by the style; and
   - Which elements are used, and which ignored, by the style.
2. The ability to scan the region from the current point to the beginning of the document (including any sub-documents), and identify the nicknames of citations defined, noting the style associated with them.
3. The ability to offer to define a new citation using an existing citation template. In proper Emacs fashion, the TAB key should perform auto-completion, or offer a list of available citation templates consistent with a partial entry. Once a template is selected, a prompt in the Emacs mini-buffer should call for each element of the citation as required.
4. The ability to provide auto-completion or a list of any citation nickname defined in the region between the current point and the beginning of the document (including sub-documents).

For novice users, this would ensure that citations are entered correctly, whatever package of blueprints is contained in `lexicite.tex`. In very long documents, it would make it easy to ensure that, in revision, nicknames are never used before they are initially defined.

BibTeX analog. Some users may prefer to maintain their citations in a separate file, as is done with BibTeX. With a little re-drafting, a similar approach could be adopted by LexiTeX, without abandoning the ability to enter citations into the document directly. Two primary changes would be required. One is an environment macro (say, `\stashlcites`) which would perform the storage operation silently (without introducing citations into the text or the tables). The other is to introduce a toggle for each nickname macro, which would indicate whether it has once been cited in the document, so that a full-form citation is produced the first time, and short-form citations thereafter.

Proxies. Some sources have long and cumbersome names. Both good taste and clarity demand that these be replaced by a shorter form in subsequent references. Foreign statute names are also often customarily referred to by an abbreviation that cannot be derived directly from the text units of a LexiTeX citation. To cover these cases, LexiTeX should provide one more option (probably =), whose usage would follow that of +, ++, +-, * and -. This nickname needs to be properly incorporated into full citations, and should replace the standard text of subsequent citations wherever appropriate.

Parallel citations. The most difficult problem still to be addressed is that of parallel citations. Law cases, in particular, often appear in more than one source. Some journals require that the specific page reference for a proposition be given for *each* source, even in subsequent abbreviated citations.

The problem is clear. In its current form, LexiTeX only knows about the six memorized citation elements, plus a single page reference. Parallel citations require that LexiTeX memorize the complete details for the remaining portions of the citation, and accept additional specific page numbers if necessary.

The syntax and programming strategy for this is in hand, but I will hold radio silence until I have a working solution to release.

Page Ranges

George Greenwade has suggested that the page specifier set in initial citations should contain an option for specifying a range, which would then be enforced in subsequent citations. Because the accuracy of legal documents is often of critical importance, this is clearly a good idea, and will be included in the next release of LexiTeX.

Ending Note

LexTeX is free of charge. The only conditions attached to its use are, first, that it be circulated as a complete package, together with its copyright and permission notices, and second, that LexTeX and its author be duly cited whenever it is used in the preparation or production of material for publication. I am always open to bug reports, and happy to receive suggestions for improvement.

I hope that LexTeX will prove useful enough to attract more law journals and legal publishers into the TeX fold. In particular, many law journals today would benefit from TeX's proficiency in typesetting foreign scripts. If this ability is combined with greater speed and certainty in the preparation of legal manuscripts, perhaps we shall see the alternatives rejected for failure to state a claim.

Standard LexTeX ver. 1.8 Citation Templates

```
\newbook{blah1}{B. Cardozo}{The Growth of The Law}{1924}+{15}.
```

↪B. Cardozo, The Growth of the Law 15 (1924).

```
\newbook{blah2}{}{Eastern Air Lines, Inc., 1978 Annual Report}{1979}+{15}.
```

↪Eastern Air Lines, Inc., 1978 Annual Report 15 (1979).

```
\newman{blah3}{Comics Magazine Ass'n of America}{Press Release No.~51}{Sept.~16, 1954}+{15}.
```

↪Comics Magazine Ass'n of America, Press Release No. 51 15 (Sept. 16, 1954).

```
\newman{blah4}{}{Telephone conversation with Douglas Borisky, Senior Revising Editor of the
\\Columbia Law Review\\}{Apr. 10, 1986}.
```

↪Telephone conversation with Douglas Borisky, Senior Revising Editor of the *Columbia Law Review* (Apr. 10, 1986).

```
\newarticle{blah5}{Cox}{Federalism and Individual Rights}{73~Nw.~U.L. Rev.}{1}{1978}+{15}.
```

↪Cox, *Federalism and Individual Rights*, 73 Nw. U.L. Rev. 1, 15 (1978).

```
\newinbook{blah6}{\\O.W. Holmes\\}{Law in Science and Science in Law}{\\in\\ Collected Legal
Papers}{210}{1920}.
```

↪O.W. Holmes, *Law in Science and Science in Law*, in Collected Legal Papers 210 (1920).

```
\newinbook{blah7}{Maitland}{The Mystery of Seisin}{\\in\\ 3 Select Essays in Anglo-American
Legal History}{591}{1909}.
```

↪Maitland, *The Mystery of Seisin*, in 3 Select Essays in Anglo-American Legal History 591 (1909).

```
\newnews{blah8}{}{Abscam Jury Sees Videotape of Deal}{San Francisco Chron., Aug.~14, 1980}
{14, col.~1}{}.
```

↪*Abscam Jury Sees Videotape of Deal*, San Francisco Chron., Aug. 14, 1980, p. 14, col. 1.

```
\newcase{blah9}{Baker v.\ Fortney}{299~S.W.2d}{563}{Mo.\ Ct.\ App.\ 1957}+{564}.
```

↪*Baker v. Fortney*, 299 S.W.2d 563, 564 (Mo. Ct. App. 1957).

```
\newecase{blah10}{Clough Mill Ltd v Martin}{[1984] 3~All ER}{982}+{986}.
```

↪*Clough Mill Ltd v Martin*, [1984] 3 All ER 982, 986.

```
\newjcase{blah11}{Decision of the Tokyo District Court, February 18,
  1988}{Hanrei jih\=o, n.~1295}
{132}+{133}.
```

↪Decision of the Tokyo District Court, February 18, 1988, Hanrei jihō, n. 1295 p. 132, 133.

```
\newstatute{blah12}{Robinson-Patman Act}{15~U.S.C.}{1982}+{{13^--13b^}\,
  {21^a^}}.
```

↪Robinson-Patman Act, 15 U.S.C. s. 13-13b, 21a (1982).

```
\newconstitutionart{blah13}{Federal Constitution}{1957}+{1}.
```

↪Federal Constitution art. 1 (1957).

```
\newconstitutionsch{blah14}{Federal Constitution}{1957}+{1}.
```

↪Federal Constitution sch. 1 (1957).

LexiTeX[a] Reference Card

$\boxed{\text{Style Template}} + \boxed{\text{Blueprint}} \longrightarrow$

$\longrightarrow \boxed{\text{Citation Template}} + \boxed{\text{Text}} \longrightarrow$

$\longrightarrow \boxed{\text{Nickname}}$ + Maybe a $\boxed{\text{Page Reference}}$

After a citation template declaration for a particular work, use the nickname to refer to that source thereafter.

The style template is called once, by `lexitex.sty` at startup. Blueprints for use by the style template are contained in `lexicite.tex`. The distribution contains blueprints for the style template which will suit most, but by no means all, purposes.

Options and Special Commands

`\lowcaseblah` before any LexiTeX citation, forces *Id.* to lowercase, should it occur.

`\lexicite`{**`blahl`**} is used to execute nicknames that include a number, such as a date.

`\\...\\` marks special typeface (usually roman) within an argument of a citation template declaration.

`\lexiproof`{**`blah`**} is used to generate a proof sheet giving most of the permutations of a nickname fired by a given citation template. This is useful when creating or editing citation templates.

* used immediately after a LexiTeX citation, suppresses output of a bare citation template declaration or nickname.

- used between a LexiTeX citation and an optional page or section reference, suppresses output of the citation template declaration or nickname, and of the page reference.

[a] LexiTeX is by Frank G. Bennett, Jr., Lecturer in Law at the School of Oriental and African Studies, University of London.
(Internet) `fbennett@clus1.ulcc.ac.uk`

+ appends a page or section number when typed after a citation template declaration or a nickname. Always enclose in braces: `+{123}`.

++ appends a plural page or section argument to any LexiTeX citation. Enclose page numbers in braces, and separate items with `\,`, `\&` or `\dash` (ranges not supported, but do no harm when used with non-statute citation templates and nicknames).

+- appends a singular page or section argument, but writes multiple references to the table file. Most often used in combination with `^...^` and `_..._` to cope with references to multiple subsections of a single statutory provision.

^ used within appended page or section references to enclose material that will appear only in the text, not in the table file. Used in all other arguments to enclose material that will appear only in initial or full-form citations.

_ used within appended page or section references to enclose material that will appear only in the table file, not in the text.

Tables

To write external table files, use:

`\makebooktable` (makes `*.btb`)
`\makearticletable` (makes `*.atb`)
`\makecasetable` (makes `*.ctb`)
`\makestatutetable` (makes `*.stb`)

Process the external table file using `makeindx` with the appropriate style file and output file name:

`lexibook.ist` → `*.bok`
`lexiarti.ist` → `*.art`
`lexicase.ist` → `*.cas`
`lexistat.ist` → `*.sta`

Place finished tables in the document using:

```
\printbooktable
\printarticletable
\printcasetable
\printstatutetable
```

Using TEX and METAFONT to Build Complicated Maps

Daniel Taupin
Université de Paris-Sud
Laboratoire de Physique des Solides
bâtiment 510
F-91405 ORSAY Cedex, France
taupin@rsovax.ups.circe.fr

Abstract

The aim of this work was to publish a catalog of all the 1500 crags and climbable rocks in France. The main source is an important TEX file describing all these places, including for each one: a serial number, a quotation of its importance, its usual name and some information about its location.

This master file is scanned by a Fortran programme which computes the coordinates of the respective marks on the paper and performs some topological analysis to settle the text labels corresponding to each mark, in order to avoid label and mark collisions. The final result is a TEX file which yields the assumedly best mark and label positions.

This TEX map has to be superimposed to a background map representing the coast lines, the main rivers and the major lakes of the country. This is done in METAFONT with a major input of *Lambert kilometric coordinates* of these features. However, the map background has to be erased at the places dedicated to TEXt labels : this is done using a set of METAFONT filtering files produced by TEX macros when compiling the labels which are "input" in the METAFONT map generating process.

Several difficulties arise, due to METAFONT limitation on numbers, and to the limited capacities of some DVI translators, particularly DVIPS.

Together with typing — and playing — organ music and typesetting statistics related to physical experiments, one of the hobbies of the author consists of rock climbing and especially maintaining a tentatively exhaustive catalog of all the 1500 known climbable crags of France outside the high mountains.

This catalog is maintained as a TEX file which has to be printed every three years, under the name "Guide des Sites Naturels d'Escalade de France". In fact the main text is a plain text which does not look very strange, of which we give a short excerpt :

29.02. ***(*) **« PLOUGASTEL »**, *SPORT*, **comm:** Plougastel-Daoulas, **Mich:** 58.4, **IGN:** 0417e; **sit:** 10 km E de Brest; **accès:** par D33 (Brest-Quimper); **héb:** camping sauvage impossible, camping perm. à Plougastel (St-Jean, en venant de Brest: sortie Landerneau, en venant de Quimper: sortie Plougastel-Daoulas, puis suivre les panneaux), camping munic. (perm.) à Brest-St-Marc (sur D33); **roch:** quartzite, grès, marbre (exc.), **climat:** pluie/neige: 1000 mm de précipitations en 150 jours/an, minimum au printemps.

29.02.1 ** **« ROC'H NIVILEN, le CUBE »**, *SPORT*, L= (1097.7, 103.2); **sit:** 1,8 km WNW de Plougastel, près du hameau de Roc'h Nivilen; **accès:** de Brest prendre D33 dir. Quimper, franchir le pont Albert Louppe sur l'Elorn, ne pas prendre la première route à D, mais la deuxième petite route, face au restaurant Ty-ar-Mor, peu avant un arrêt de bus (ne pas aller jusqu'à l'embranchement vers Plougastel) et la suivre W sur env. 1km d'où un sentier conduit au Cube; pour Roc'h Nivilen, continuer jusqu'à quelques maisons pour prendre S un chemin qui conduit à l'W de Roc'h Nivilen et revient par l'W (ne pas aller directement du Cube à Roc'h Nivilen: privé); **propr:** commune + privé; **roch:** quartzite/marbre (exc.), 20 voies, 18-30 m, 2–6b, souvent équipées; **biblio:** *TOPO-GUIDE* (CAF Brest, 1984); **obs:** on y trouve des voies faciles ou peu soutenues en exc. rocher; site très fréquenté; **ATTENTION:** bien que moins spectaculaire qu'à Pen-Hir, la corrosion des pitons y est importante: certains pitons sont enduits au minium, mais la partie cachée n'est pas protégée, il faut donc impérativement renforcer l'assurance avec des coinceurs.

29.02.2 *** **« L'IMPÉRATRICE »**, *SPORT*, L= (1098.5, 103.5); **sit:** au S de l'Elorn, 2 km E du pont de l'Elorn (route Brest-Quimper); **accès:** de Brest dir. Quimper, sortie Plougastel, redescendre vers Brest sur 300 m, tourner à D vers Le Passage, les rochers sont visibles au niveau d'un transformateur, NE PAS STATIONNER à proximité des rochers, mais continuer jusqu'à Le Passage, vaste parking au bord de l'Elorn; revenir à pied aux rochers par la route; **propr:** privés; **roch:** grès quartzite (exc.), 62 voies, 30-45 m, 3b–6b; **biblio:** Alpi. Rando. nov. 1979, *TOPO-GUIDE* (CAF Brest, 1984); **obs:** face N, gras par temps humide, plus difficile en moyenne que Roc'h Nivilen; équipement inox et rapproché dans des voies d'initiation en face S, pour le reste équipement médiocre à pourri et espacé (intervalle moyen 5-6 m).

29.03. ** « OUESSANT, CRÉAC'H », *BLOCS/AVENT.*, **comm:** Lampaul, **Mich:** 58.2, **IGN:** 0317w, L= (1112, 50); **sit:** aiguilles et chaos rocheux en bord de mer, au NW de l'île, à la pointe de Pern, et le long de la côte NW de l'île entre le phare de Créac'h et Pors Yusin (4 km de côte rocheuse); **accès:** par bateau depuis Brest ou le Conquet, puis 4 à 8 km à pied ou en vélo; **héb:** camping (perm.); **roch:** granite (exc., lichen dans les zones non balayées par la mer), 10-20 m; **obs:** risques de marées noires; quelques autres rochers grimpables sur la côte S entre Pors Alan et Kergoff.

The corresponding source TeX looks like :

```
\newsper{29.02.}{***(*)}{\ecp}{PLOUGASTEL}
\cm Plougastel-Daoulas, \m 58.4, \ign 0417e;
\sit 10~km E de Brest; \ac par D33
(Brest-Quimper); \heb camping sauvage
impossible, camping perm. \'a Plougastel
(St-Jean, en venant de Brest: sortie
Landerneau, en venant de
.....
```

The Requirements for Maps

Obviously — and even for people who do not read fluently the French language — such a catalog is of poor interest if there is not a set of maps to help the reader find the spots.

Several means of printing maps can be thought of. The first obvious one consists in drawing the map with a pencil and sending it to the publishing company together with the TeX text, either on diskettes or as camera ready papers. The obvious drawback is that each update of the guide — four editions (Taupin, 1982, 1984, 1986, and 1989) were published since 1981 and the fifth is being prepared — requires a complete remake of the whole of the drawings, which is not a very efficient method.

Another way of doing it would consist of making the drawings using a computer and some drawing softwares like MacDraw, Microsoft Paint or Free Hand, or in scanning a hand made picture to produce some picture file. The difficulty resides in the fact that :

- Picture files have a great number of formats : TIFF, MSP, PCX, PCL, etc., and of course EPSF.
- Pictures must be partially erased to give precedence to *labels* indicating the names of the spots and of the towns.
- The final printable file should be sent to various publishing companies and local printing devices whose "language" could be Postscript as well as company specific codings like PCL.
- Picture maps should be automatically regenerated each time a new site is included — or cancelled — in the master file, without having to rebuild the whole of a map when only one square centimeter has been modified.

Unfortunately, the available `dvips` are somewhat "Postscript addicted" and cannot accept other picture formats, while the `emTeX` related drivers can insert a lot of things... except Postscript !

Secondly, notwithstanding several messages sent to the TeX addicted gurus, we did not find any software able to erase selectively some areas of any picture file.

The consequence was that the only way of having *portable* pictures was to draw them using METAFONT.

The Mapping Process

Extracting site names and locations. The first thing to do is to scan the megabyte of master TeX file(s) to extract the first names of the sites (e.g., `PLOUGASTEL`, `ROC'H NIVILEN`) together with their location, i.e., something like `L= (1098.5, 103.5)` (case of known kilometric *Lambert* coordinates) or `10~km E de Brest` (case of an approximate relative description). If the situation is given as a distance and an orientation from a known town, a catalog of famous towns is searched and the coordinates are easily computed.

Of course this could be done in TeX language but we preferred to do it faster using usual programming languages (Fortran in this case) able to handle *real* numbers. The resulting file looks like:

```
S=  1 3 2902.00  Plougastel
L= ( 697.50,  103.50)
S=  2 2 2902.01  Roc'h Nivilen
L= ( 697.70,  103.20)
S=  2 3 2902.02  Imp&ratrice
L= ( 698.50,  103.50)
S=  0 2 2903.00  Ouessant
L= ( 712.00,   50.00)
```

Building the map of site and town labels. Once the above file — 1500 entries in fact — has been built, we are faced with the hardest problem, namely computing the positions of all the marks and labels (sites and towns) in each of the 38 maps (an arbitrary number) so that all — or nearly all — sites have their label and their mark in the relevant maps of various scales, avoiding overlapping labels and labels colliding with site and town marks.

This is actually a tough job of trial and error geometry, definitely impossible in TEX because of its use of heavy floating point arithmetic. We did it in Fortran — which offers a safer encapsulation of variables — but it could have been done in any language like Pascal or C. Since the process is not written in TEX it is not worth details here; we just mention that it takes four hours on a 386 with math coprocessor, and a long night on an average 286.

The result, however, is a series of TEX files — one per map — which looks like the following :

```
\lochead{Bretagne}\rx
\maperaseopen{BRETAMAP.mfc}
\newzone{15.000cm}{22.200cm}{.0714}\relax
\global\rthick= .40pt\rx
\guidepcmap{17}\rx
\hpoint{13.523cm}{5.023cm}{\threestar}\rx
\hpoint{14.109cm}{2.084cm}{\twostar }\rx
\hpoint{14.921cm}{16.221cm}{\onestar }\rx
....
\hpoint{3.425cm}{14.818cm}{\mkville }\rx
\Hpoint{2.422cm}{14.893cm}{\bnw
   {.988cm}{.269cm}{\ftv Brest}}\rx
\hpoint{9.211cm}{17.568cm}{\mkville }\rx
\Hpoint{6.992cm}{17.411cm}{\bsw
   {2.204cm}{.269cm}{\ftv Perros-Guirec}}\rx
\hpoint{12.496cm}{15.068cm}{\mkville }\rx
\Hpoint{12.661cm}{15.143cm}{\bne
   {1.596cm}{.269cm}{\ftv St-Brieuc}}\rx
\hpoint{12.211cm}{8.496cm}{\mkville }\rx
\Hpoint{12.376cm}{8.571cm}{\bne
   {1.140cm}{.269cm}{\ftv Vannes}}\rx
....
\Hpoint{4.251cm}{14.821cm}{\Bne
   {2.677cm}{.366cm}{\ftc 2902 Plougastel}}\rx
\Hpoint{.394cm}{15.606cm}{\bse
   {2.160cm}{.269cm}{\ftb 2903 Ouessant}}\rx
\Hpoint{.063cm}{13.389cm}{\bsw
  {2.800cm}{.269cm}{\ftb Pte de Dinan 2904}}\rx
....
\Hpoint{.200cm}{.200cm}{\cartouche{5.171cm}
  {1.500cm}{3.571cm}{17: Bretagne}{10}}\rx
\hpoint{.000cm}{.000cm}{\vrule width\rthick
  height 22.200cm}\rx
\hpoint{15.000cm}{.000cm}{\vrule width\rthick
  height 16.201cm}\rx
....
\maperaseclose
```

Most of the macros invoked in this text are `\hpoint` or `\Hpoint` (a boxed variant) which just shift and raise the \hbox containing the text of the third argument by the first two. This method was described ten years ago by D. Knuth (*The TEXbook*, page 289) in a chapter named "Dirty Tricks" and the macros \bne, \bsw, etc., just adapt the shape of the label to the case where it is at the northeast of the mark, at the southwest, etc.

What is more interesting is the presence of the macros `\maperaseopen` and `\maperaseclose` which open a file (here `BRETAMAP.mfc`) which will contain METAFONT macro calls. In fact each of these macros like `\bne` writes its paper coordinates (the first two arguments) and its TEX generated box size into the *map erase file* in the form:

```
mapcancel(2.422cm,14.893cm,6.944pt,0pt,28.112pt)
mapcancel(6.992cm,17.411cm,7.653pt,0pt,62.709pt)
mapcancel(12.661cm,15.143cm,6.944pt,0pt,39.028pt)
mapcancel(12.376cm,8.571cm,6.944pt,0pt,29.806pt)
```

Thus, provided it is `input` by METAFONT, the `BRETAMAP.mfc` will tell which rectangles of the background map have to be erased when generating the map of the region (Britanny in this case).

Generating the background maps. Several problems have to be dealt with:

1. getting the x, y coordinates of the geographic features, adapted to each map scale and origins;
2. drawing hashed lines to represent water areas within sea and lake contours;
3. erasing the places dedicated to spot name labels.

Getting the coordinates of the geographic features. The x, y coordinates must obviously be entered manually, and for example the description of the river Seine contains METAFONT statements like:

```
Writeline(803, 485, ese); % Pont de Brotone
Writeline(790, 489, ese); %
Writeline(790, 493, ne); % Pont d'Yville
Writeline(799, 494, right); % Duclair
Writeline(791.5, 496, ssw); %
Writeline(784, 498, right); % La Bouille
Writeline(785, 502, nne); % Grand Couronne
Writeline(794, 505, ene); % Canteleu
Writeline(794, 511, ese); % Rouen
Writeline(783, 509, wsw); % Oissel
....
```

and the Atlantic side is similar:

```
writeline(968, 597); % Dunkerque
writeline(959.8, 561.920); % Calais
writeline(949.9, 543.647); % Gris-Nez
writeline(934, 542); % Boulogne/M.
writeline(887, 540); % Baie de somme
writeline(876, 550); % Baie de somme
writeline(877, 540); % Baie de somme
writeline(844, 507); % Dieppe
....
```

The `writeline` and `Writeline` METAFONT macros differ by the fact that one concatenates the points to the current path as broken lines while the other assumes Bezier curves with stated orientation of the path at the given point. Once all points of a given feature, for instance the river Seine, have been accumulated in a METAFONT path variable, it has just to be drawn with the convenient pen.

Drawing hashed water areas. This is slightly more difficult. First, and if it is not an inland lake, the contour has to be completed so that a `fill...cycle` is not meaningless. Secondly, this filling should not be a full black area but an area covered with parallel equidistant lines. This is simply done by :

- `fill`ing inside all the wanted contours in black;
- `cull`ing the picture to ones for safety;
- superimposing to the whole of the picture a set of parallel lines, regardless of the contour; assuming xxa, xxb, yya and yyb to be paper map boundaries, this done by:

```
for k=yya step 0.1in until yyb:
  draw (xxa,k)--(xxb,k);
endfor;
```

- `cull`ing the picture to values ≥ 2; thus only the spots at mathematical intersection of the parallel lines and the filled area are kept; this is a very simple METAFONT command, namely:

```
cull currentpicture dropping (-infinity,1);
```

Obviously, the coast and river lines have to be `draw`n *after* the hashing of the water areas.

Erasing the place for town and site labels. This is simply done by some METAFONT command like `input BRETAMAP.mfc`; then the `mapcancel` command is rather simple, namely:

```
def
mapcancel(expr xxm, yym, htrec, dprec, wdrec)=
  numeric yybas, yyhaut, xxgauche, xxdroit;
  yybas:=yym-dprec;
  yyhaut:=yym+htrec;
  xxgauche:=xxm;
  xxdroit:=xxm+wdrec;
  cullit;
  unfill (xxgauche,yybas)--(xxgauche,yyhaut)--
    (xxdroit,yyhaut)--(xxdroit,yybas)--cycle;
enddef;
```

Generating the complete maps. This just requires the correct insertion of the backround font characters in the `guidepcmap` referenced as the fifth line of the TEX source previously given. Two excerpts of the results are given in the last two (one column) pages of this paper.

The Difficulties

As previously explained, map designing under TEX+METAFONT is easy... in principle. In fact the difficulties do not come from TEX nor METAFONT fundamentals, but they come from the actual limitations of both METAFONT and the printing drivers.

The METAFONT difficulties. METAFONT appears to be a powerful language to handle *pictures* but it exhibits some drastic limitations in the handling of *real* numbers. In fact, regardless of the computer, it has no *floating point* arithmetic, but it manipulates `numeric` values in *fixed point* representation, with a maximum of 4095.9999. Thus, kilometric coordinates within European countries are entered without problems, but problems happen when map scaling wants to convert a latitude of 800 km to paper coordinates under a scale of 1:100 000.

Of course such a point will fall out of the sheet, but the mere computation of its paper coordinate makes METAFONT complain about "overflow", stop, and restart with unpredictable results when the diagnostic is ignored.

Since we cannot have ten versions of the path of the river Seine fit for every possible scale, the `writeline` macro must check the input parameters against the scale and skip the point *before* an overflow can occur.

Other problems occur with water areas. In fact a small error in the recording of coast points can lead the water contour to look like an "8" rather than simple convex contour; in that case, METAFONT has difficulties in finding what is the internal part of the "8" and what is outside, and this often results in... emptying the sea, a thing which was not really planned by the Creator!

On the other hand, METAFONT has no problems creating characters of size 15cm×22cm, at least in the 300 DPI resolution. Thus, our first attempt consisted in making one character for each or the 38 maps, with the pleasant consequence of having only one font for all our maps.

The driver problems.
Previewing on MS-DOS computers. Our first attempts with $15 \times 22\text{cm}^2$ characters were surprisingly displayed without difficulty although slowly by Eberhardt MATTES' `dviscr`.
Printing in PCL language (Laserjet). Problems began with `dvihplj` which refused to print some maps but not the others. After investigation we found that the problem disappeared if we carefully erased (using the `cull` command) *all possible points* outside the actual map. Then we were able to print

the full set of maps on various HP-Laserjet printers using `dvihplj`.

Postcript printing. Serious problems were encountered when trying to convert the DVI to Postscript. We used the `dvips` of best reputation, namely the one created (and maintained) by Tomas Rokicki (with Radical Eye Software copyright) which is available in C on various `ftp` servers.

We discovered first that many versions of this `dvips` crashed with the diagnostic "`No memory`" when run on 286 versions. Obviously several 386 versions do exist, but their behaviour strongly depends on the C compiler used to build the executable programme, and on the fact the configuration takes advantage or not of the 386 memory extension facilities. Half a dozen were tried — for that purpose but also to convert other papers with many fonts — and the most powerful seems to be the version recently provided by F. Popineau (École Supérieure d'Électricité, Metz, France), which was compiled with *Metaware High C* using the *PharLap* DOS extender.

In fine, we crashed against a repetitive diagnostic, independent of the total number of fonts in the DVI, which was:

```
DVIPS.EXE: Cannot allocate more than 64K!
```

After some trials, we understood that this was a drastic limit of `dvips` not depending on the total size of the font or the PK file but of the size of the characters. Thus we had to change all the METAFONT scheme in order to divide each maps into a number of characters so that they do not exceed (approximately) the square inch. Obviously, the total number of characters was multiplied by a factor of 20 (in the present version at least) so that the maps had to be shared into several fonts, which in turn resulted in a more sophisticated TEXing of the maps.

Of course a logical solution would have been to use one font per map (as does BM2FONT) but at the other extreme, the number of fonts in a DVI is also hardly limited in both TEX and the drivers!

Up to now, this unsmart way of cutting maps seems to work and the Postscript file seems to be accepted by most (not all) Postscript printers we have access to. But the question of the conformity of all Postscript printers is another problem...

A conclusion METAFONT and the `dvi`-to-xxx drivers are intended to manipulate text and characters of a reasonable size, typically the square inch. But building maps strongly exceeds the typical sizes of characters and distance conversions exceed METAFONT's handling of numeric values.

A great deal of possible solutions to this problem can be thought of, for example permitting TEX to handle thousands of fonts. We think however that the simplest should be to have just two increased dimensions:

1. Eight byte handling of numerics in METAFONT, still in fixed point for compatibility with previous versions.
2. `dvips` drivers able to handle huge characters — as the `dvihplj` by Eberhardt Mattes — in the hope that this is not due to a Postscript hard limitation.

Some Examples

Two maps are presented in this paper:[1]

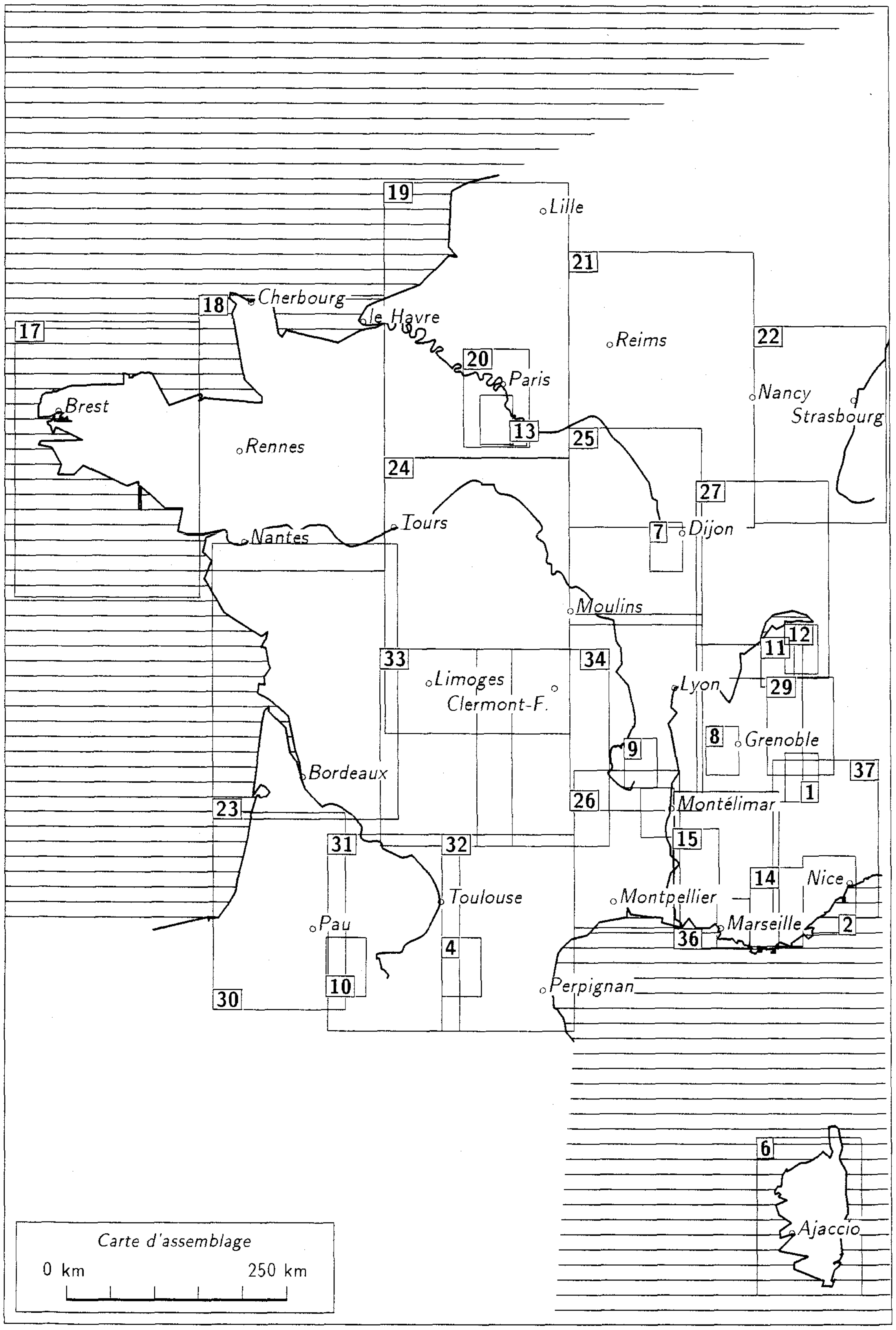
1. A general of France showing the layout of the 37 regional maps, with seas and main rivers drawn by METAFONT as previously explained, and towns and caption built in TEX.
2. A regional map of Britanny showing some sea areas, the river Loire and a number of site labels possibly shifted (Puits de la Roche) to avoid collisions.

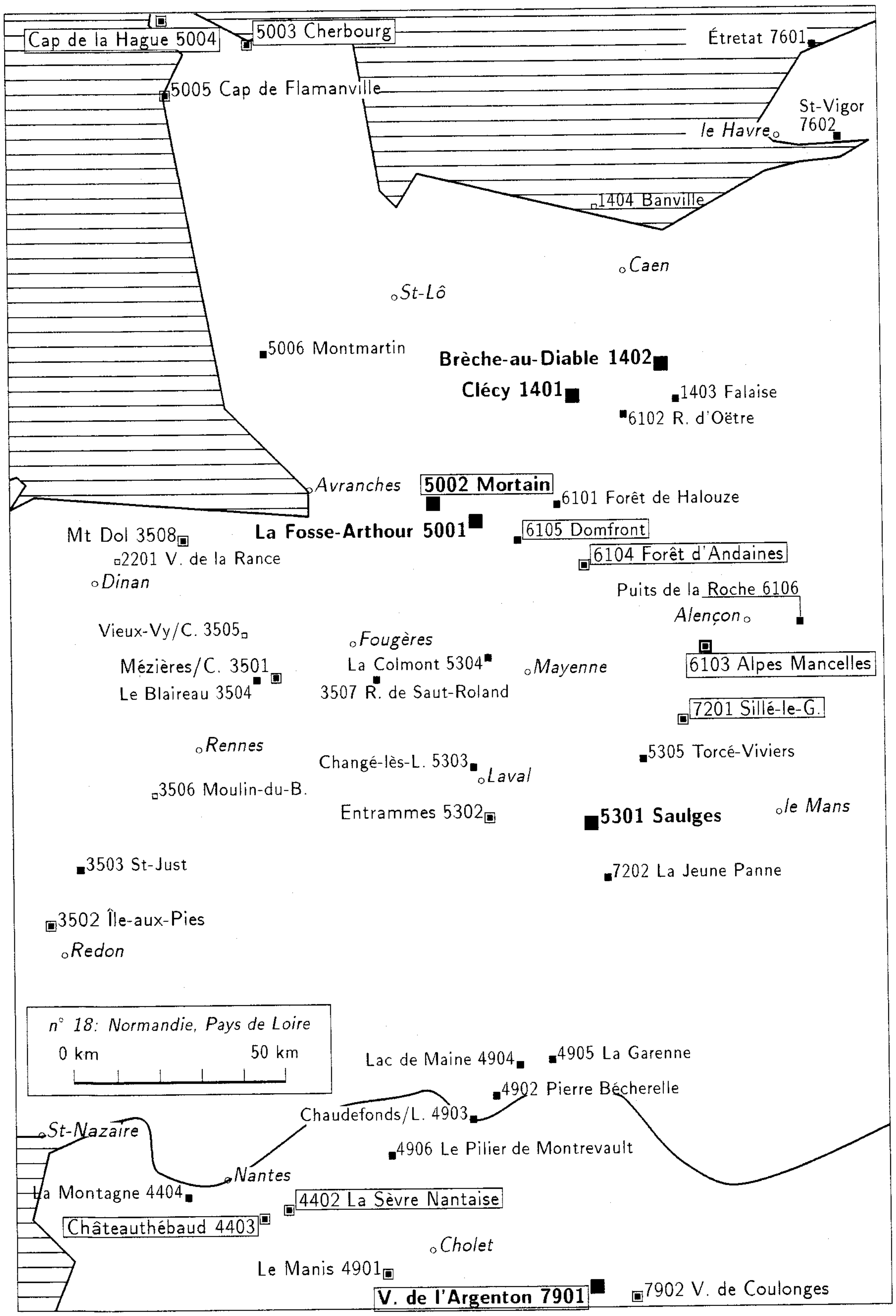
Bibliography

Daniel Taupin, "Guide des Sites Naturels d'Escalade de France", *COSIROC*, Paris, 1982, 1984, 1986, 1989.

Donald E. Knuth, *The TEXbook*, Addison-Wesley Publishing Company, 1984.

[1] *Editors' Note:* It proved impossible to generate the map fonts at the full resolution of the typesetter, due to METAFONT limitations. The examples are therefore set at only 300 dpi.


19
Lille
21
18
Cherbourg
le Havre
17
Reims
22
20
Paris
Nancy
Strasbourg
Brest
13
25
Rennes
24
27
Tours
7
Dijon
Nantes
Moulins
12
11
33
Limoges
34
Lyon
29
Clermont-F.
8
Grenoble
9
37
Bordeaux
1
26
Montélimar
23
31
32
15
14
Nice
Toulouse
Montpellier
Pau
Marseille
2
36
4
10
Perpignan
30
6
Ajaccio
Carte d'assemblage
0 km
250 km


Cap de la Hague 5004
5003 Cherbourg
Étretat 7601
5005 Cap de Flamanville
St-Vigor 7602
le Havre
1404 Banville
Caen
St-Lô
5006 Montmartin
Brèche-au-Diable 1402
Clécy 1401
1403 Falaise
6102 R. d'Oëtre
Avranches
5002 Mortain
6101 Forêt de Halouze
Mt Dol 3508
La Fosse-Arthour 5001
6105 Domfront
6104 Forêt d'Andaines
2201 V. de la Rance
Dinan
Puits de la Roche 6106
Alençon
Vieux-Vy/C. 3505
Fougères
Mézières/C. 3501
La Colmont 5304
Mayenne
6103 Alpes Mancelles
Le Blaireau 3504
3507 R. de Saut-Roland
7201 Sillé-le-G.
Rennes
Changé-lès-L. 5303
Laval
5305 Torcé-Viviers
3506 Moulin-du-B.
Entrammes 5302
5301 Saulges
le Mans
3503 St-Just
7202 La Jeune Panne
3502 Île-aux-Pies
Redon
n° 18: Normandie, Pays de Loire
0 km
50 km
Lac de Maine 4904
4905 La Garenne
4902 Pierre Bécherelle
Chaudefonds/L. 4903
St-Nazaire
4906 Le Pilier de Montrevault
Nantes
La Montagne 4404
4402 La Sèvre Nantaise
Châteauthébaud 4403
Cholet
Le Manis 4901
V. de l'Argenton 7901
7902 V. de Coulonges

MusicTEX: Using TEX to Write Polyphonic or Instrumental Music

Daniel Taupin
Université de Paris-Sud
Laboratoire de Physique des Solides
bâtiment 510
F-91405 ORSAY Cedex, France
taupin@rsovax.ups.circe.fr

Abstract

MusicTEX is a set of TEX or LATEX macros — initially posted three years ago and now used by dozens of music typesetters — which are fit to typeset polyphonic, instrumental or orchestral music. It is able to handle an important number of instruments or voices (up to nine) and staffs (up to four for each instrument). Many of the usual ornaments have been provided, including several note sizes which can handle grace notes or extra music like cadenzas.

A recent enhancement consisted of providing a facility for several staff sizes in the same score, thus enabling full size staffs to smaller "reminding" staffs.

The LATEX version is not really fit for printing full scores but it has be used to produce musicographic texts including many (small but numerous) music excerpts.

Except for the risk of typing errors due to a sophisticated set of macros, the major difficulty still resides in glue and line breaking in the case of irregular music and slurs.

What is MusicTEX?

Several packages exist which provide the personal computer addict a facility for typesetting music. For instance we saw examples from *Personal Composer* and *Musictime*, and we experimented first with the MuTEX package (see Steinbach and Schofer, 1987, 1988), the latter being based on TEX and METAFONT. However, all these packages have limitations: either the output quality (*Personal Composer*) or the complexity of the score (*Musictime*), or the number of staffs (MuTEX).

Thus, a few years ago, we could not resist the temptation of building a new package — in fact a set of TEX macros and fonts — which would be able to typeset complex polyphonic, orchestral or instrumental music. In fact our primary intention was to extend MuTEX to several staffs, but was quickly apparent that rewriting the whole of the macros was a better solution and we only used MuTEX's METAFONT code as a starting point.

Although not perfect, MusicTEX appears to be a powerful tool which can handle up to nine distinct instruments, each having from zero (for lyrics) to four staffs. Of course it can handle chords or polyphonic note settings in the same staff and we have used it to typeset realistic music for choirs and instruments, including organ.

It must be emphasized that MusicTEX is not intended to be a compiler which would translate into TEX some standard musical notations, nor to decide by itself about aesthetic problems in music typing. MusicTEX only typesets staves, notes, chords, beams, slurs and ornaments as requested by the engraver. Since it makes very few typesetting decisions, MusicTEX appears to be a versatile and rather powerful tool, but in turn it should be interfaced by some precompiler for the engraver who wants aesthetic decisions to be automatically made by somebody (or something) else.

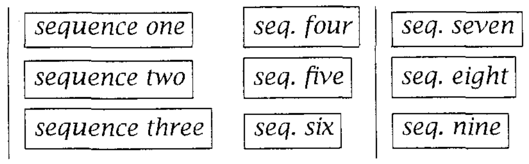
One can also mention a secondary use of MusicTEX as a *target* language for music coding, namely the MIDI2TeX package by Hans Kuykens, which translates MIDI data files into MusicTEX source code (Kuykens, 1991). Notwithstanding capacity problems, a LATEX style has also been provided (it was used to typeset the present paper) but this `musictex` style is fit for musicographic books rather than for normal scores to be actually played.

MusicTEX principal features

Music typesetting is two-dimensional. Most of the people who just learned a bit of music at college probably think that music is a linear sequence of symbols, just as literary texts to be TEX-ed. In fact, with the exception of strictly monodic instruments like most orchestral wind instruments and solo voices, one should be aware that reading music is actually a matricial operation: a musician playing a chordal instrument — guitar, piano, organ — or looking at more than one staff — a choir singer, a conductor — successively reads *columns* of simultaneous notes which he or she plays or at least watches in order to be in time with the others.

In fact, our personal experience of playing piano and organ as well as sometimes helping as an alternate Kapellmeister leads us to think that actual music reading and composing is a slightly more complicated intellectual process: music reading, music composing and music thinking seems to be a three-layer process. The musician usually reads or thinks several consecutive notes (typically a long *beat* or a group of logically connected notes), then he goes down to the next instrument or voice and finally assembles the whole to build a part of the music lasting roughly a few seconds. Then he handles the next *beat* or *bar* of his score.

Thus, it appears that the most logical way of coding music consists of horizontally accumulating a set of *vertical combs* with *horizontal teeth* as described below:



This is the reason why the fundamental *macro* of MusicTEX is of the form

```
\notes ... & ... & ... \enotes
```

where the character & is used to separate the notes (or the groups of notes) to be typeset on the respective staffs of the various instruments, starting from the bottom.

In the case of an instrument whose score has to be written with several staffs, these staffs are separated by the character |. Thus, a score written for a keyboard instrument and a monodic or single staff instrument (for example piano and violin) will be coded as follows:

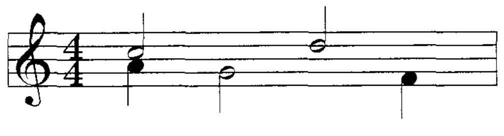
```
\notes ... | ... & ... \enotes
```

for each column of simultaneous *groups of notes*. It is worth emphasizing that we actually said *"groups of notes"*: this means that in each section of the previous macro, the music typesetter is welcome to insert not only chord notes to be played at once, but small sequences of consecutive notes which build something he understands as a musical phrase. This is why note typing macros are of two kinds in MusicTEX, namely the note macros which are not followed by spacing afterwards, and those which induce *horizontal* spacing afterwards.

The spacing of the notes. It seems that many books have dealt with this problem. Although it can lead to interesting algorithms, we think it is in practice a rather minor one.

In fact, each column of notes does not necessarily have the same spacing and, in principle, this *spacing* should depend on the shortest duration of the simultaneous notes. But this cannot be established as a rule, for at least two reasons:

1. spacing does not depend only on the local notes, but also on the context, at least in the same bar.
2. in the case of polyphonic music, exceptions can easily be found. Here is an example:



where it can be clearly seen that the half notes at beats 2 and 3 must be spaced as if they were quarter notes since they overlap, which is obvious only because of the presence of the indication of the *meter* 4/4.

Therefore, we preferred providing the engraver with a set of macros having specific spacings (`\noteskip`) whose ratio to a general basic spatial unit `\elemskip` increases by a factor of $\sqrt{2}$ (incidentally, this can be adjusted):

```
\notes ... & ... & ... \enotes   %
          1 basic spatial unit
\Notes ... & ... & ... \enotes   %
          1.4 basic spatial units
\NOtes ... & ... & ... \enotes   %
          2 basic spatial units
\NOTes ... & ... & ... \enotes   %
          2.8 basic spatial units
\NOTEs ... & ... & ... \enotes   %
          4 basic spatial units
\NOTES ... & ... & ... \enotes   %
          5.6 basic spatial units
```

The size of both the basic spatial unit (`\elemskip`) and the note-specific spacing (`\noteskip`) can be freely adjusted since they are not `\global`. In addition, MusicTEX provides a means of adjusting the basic spacing `\elemskip` according to an average number of elementary spaces within a line (macro `\autolines`).

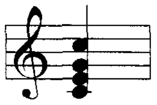
Music tokens, rather than a readymade generator. The tokens provided by MusicTEX are:

- the note symbols *without stems*;
- the note symbols *with stems, and hooks for eighth notes and beyond*;
- the indications of beam beginnings and beam ends;
- the indications of beginnings and ends of ties and slurs;
- the indications of accidentals;
- the ornaments: arpeggios, trills, mordents, pincés, turns, staccatos and pizzicatos, fermatas;
- the bars, the meter and signature changes, etc.

As an example, a half note of pitch *A* (the *A* at the top of the bass clef) with stem up is coded as `\hu a` and all pitches above that *A* are represented with lowercase letters up to `z`; uppercase letters represent grave notes, i.e.; those usually written under the bass clef. In the same way `\wh h` produces an *A* (the one in the middle of the G clef staff, i.e., 445 Hz approx.) whose duration is a *whole note*, `\qu c` produces a *C* (250 Hz approx.) whose value is a *quarter note with stem up*, `\cl J` produces a *C* (125 Hz approx.) whose duration is an *eighth note with stem down*, etc.

It is worth pointing out that pitch coding in MusicTEX is related to the actual note pitch, not to the note head position under a given clef. Thus, if the typesetter wants to change the active clef of a part of the score, he doesn't have to change the pitch codings, perhaps only the sense of the stems and of the beams.

To generate quarter, eighth, sixteenth, etc. chords, the macro `\zq` can be used: it produces a quarter note head whose position is memorized and recalled when another stemmed note (possibly with a hook) is coded; then the stem is adjusted to link all simultaneous notes. Thus, the perfect C-major chord, i.e.,

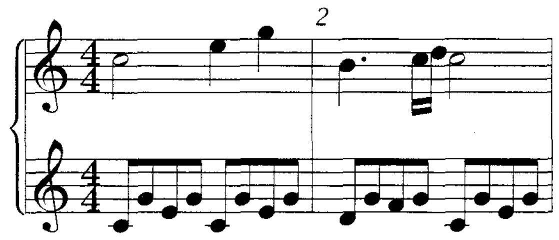


is coded `\zq c\zq e\zq g\qu j` or, in a more concise way, `\zq{ceg}\qu j` (stem up): in fact, single notes are treated... like one-note chords.

Beams. *Beams* are generated using macros which define their beginning (at the current horizontal position), together with their altitude, their sense (upper or lower), their multiplicity, their slope and their reference number. This latter feature — the reference number — appears to be necessary, since one may want to write beams whose horizontal extents overlap: therefore, it is necessary to specify which beam the notes hang on and which beam is terminated at a given position.

Setting anything on the score. A general macro (`\zcharnote`) provides a means of putting any sequence of symbols (in fact, some `\hbox{...}`) at any pitch of any staff of any instrument. Thus, any symbol defined in a font (letters, math symbols, etc.) can be used to typeset music.

A simple example. Before entering other details, we give below an example of the two first bars of the sonata in C-major K545 by MOZART:



The *coding* is set as follows:

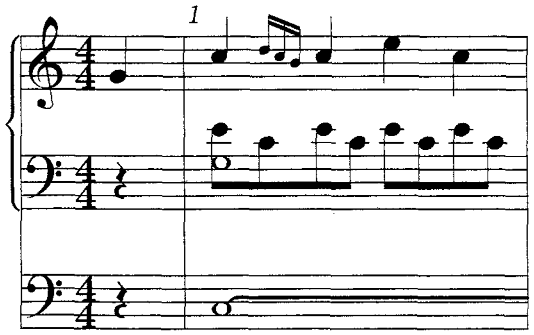
```
\begin{music}
\def\nbinstruments{1}\relax % single instrument
\nbporteesi=2\relax         % with two staffs
\generalmeter{\meterfrac44}\relax % 4/4 meter
\debutextrait               % start real score
\etroit                     % 10pt note spacing
\temps
\Notes\ibu0f0\qh0{cge}\tbu0\qh0g|\hl j\enotes
\temps
\Notes\ibu0f0\qh0{cge}\tbu0\qh0g\relax
                         |\ql l\sk\ql n\enotes
\barre                      % bar
\Notes\ibu0f0\qh0{dgf}|\qlp i\enotes
\notes\tbu0\qh0g|\ibbl1j3\qb1j\tbl1\qb1k\enotes
\temps
\Notes\ibu0f0\qh0{cge}\tbu0\qh0g|\hl j\enotes
\finextrait                 % end excerpt
\end{music}
```

- `\ibu0f0` begins an upper beam, aligned on the *f*, reference number 0, slope 0.
- `\tbu0` terminates this beam before writing the second *g* by means of `\qh0g`.
- `\qh..` indicates a note hanging on a beam.
- `\sk` sets a space between the two quarters at the right hand, so that the second is aligned with the third eighth of the left hand.
- `\qlp` is a quarter with a point and stem down.
- `\ibbl1j3` begins a double beam, aligned on the *C* (`j` at this pitch) of slope 0.15.

Signatures. Signatures are usually stated for all instruments, such as: `\generalsignature=-2` which sets two flats on each staff; however this global signature can be partly overridden by instrument specific statements such as: \signatureii=1 which puts one sharp on the staffs of *instrument number* 2 (ii). Of course, the current signature may change at any time as well as meters and clefs.

Transposition. Provided some precaution is taken concerning the accidentals, MusicTEX can transpose or partly transpose a score. In fact, there is an internal register named \transpose, the default value of which is zero, but which may be set to any reasonable positive or negative value. Then, it offsets all symbols pitched with letter symbols by that number of pitch steps. However, it will neither change the signature nor the local accidentals, and if, for example, you transpose by 1 pitch a piece written in *C*, MusicTEX will not know whether you want it in *D*♭, in *D* or in *D*♯. This might become tricky if accidentals occur within the piece, which might have to be converted into flats, naturals, sharps or double sharps, depending on the new chosen signature. To circumvent this trouble, *relative* accidentals have been implemented, the actual output of which depend on the pitch of this accidental and of the current signature.

Grace notes and cadenzas. In addition to its facility for generating either sixteen point or twenty point staffs with note heads of corresponding size, MusicTEX also allows the user to type smaller notes, in order to represent either *grace notes*, *cadenzas* or a proposed realization of a *figured bass*. This may give something like:



Selecting special instrument scores. A frequent question is: *"Can I write an orchestral score and extract the separate scores for individual instruments?"* The answer is 95% *yes*: in fact, you can define your own macros \mynotes...\enotes, \myNotes...\enotes with as many arguments as there are in the orchestral score (one hopes this is less than or equal to 9, but TEXperts know how to work around it) and change their definition depending on the selected instrument (or insert a test on the value of some selection register). But the limitation is that the numbering of instruments may change, so that \signatureiii may have to become \signaturei if instrument *iii* is alone. But, in turn, this is not a serious problem for average TEX wizard apprentices.

How to get it

The whole *distribution* fits on a single 1.2Mbyte or 1.44Mbyte diskette. It is also available on anonymous ftp server rsovax.ups.circe.fr (130.84.128.100), after selecting the subdirectory [anonymous.musictex]. Several other ftp sites also provide it, especially ftp.tex.ac.uk and ftp.gmd.de. All sources are provided, including fonts. It can also be automatically e-mailed (uuencoded) by means of the message SENDME MUSICTEX sent to FILESERV@SHSU.BITNET.

Implementation

The macro file MusicTEX contains approximately 2500 lines of code, approximately 80 000 bytes. This requires your score to be compiled by the most extended versions of TEX (65 000 words of working memory), or with "BigTEX" processors which are unfortunately slow on 286 PCs, due to a great deal of disk input/output.

In particular, notwithstanding the fact that a great many dimension registers have been moved to \fontdimen registers — an ugly but efficient way of doing it — the number of registers it uses can hinder its compatibility with some LATEX styles or with LATEX itself in case of restricted memory availability.

Recent easy enhancements. Many enhancements have been asked for, and this is proof that MusicTEX is considered useful by many people. Some of these enhancements which seemed hard were in fact rather easy to implement, for example, small notes to represent grace notes and cadenzas or narrow staffs to represent informational score — not to be played — like the violin part above the actual piano staffs to be played by the reader. But others may induce heavy problems, for example, the need of having *nice* slurs and ties.

The tie/slur problem. While typesetting notes and even beams is a rather simple problem because it is *local typesetting*, ties and slurs are much more difficult to handle.

Or course there is little problem in the case of a typesetter wanting a slur or a tie binding two consecutive notes, not separated by a bar. In practice this *very restricted* use of slurs or ties can easily be solved by putting some symbols extracted from the `slur16` or `slurn16/slurn20` fonts somewhere on the staffs using the general use `\zcharnote` macro.

But serious music engravers know that many ties are supposed to link notes which are on both sides of a bar, which is a likely place to insert line breaks, so that the *tie* coding must have various versions and sizes to resist that possible line breaking. What has been said about ties is still more serious in the case of *phrasing slurs* which may extend over several bars, lines and sometimes pages. In this case, their shape is not only a question of producing a long curved symbol of nice looking shape, it also has to cope with *glue*. Unfortunately, the way of typing music does not accept *ragged lines* but equal length lines, even for the last line of a music piece. Thus, long distance slurs and ties need to be cut into separate parts (beginning, continuing(s), endings) which TEX can only link using *horizontal line overlaps* or `\leaders` to insure slur continuity over this unavoidable glue.

Therefore, up to now, ties and slurs have been implemented in a way which may look rather ugly, but we think it is the only way of implementing *in one pass* ties and slurs which run *across glue*. The principle is to have tie/slur symbols with a rather long part of horizontal stuff. Then, each time glue occurs and each time a group of notes is coded while a slur or tie is pending, an `\hrule` is issued which overlaps or links to the preceeding tie/slur symbol so that the final output seems to contain a continuous line. Unfortunately, this is possible only in the glue expansion direction, namely in the horizontal direction.

A recent enhancement consisted of providing two kinds of slur macros (`\ilegun`*p* and `\Ilegun`*p*, same for lower slurs) to have variable size initial and final curved slur symbols which the user can choose according to his intention to have short or long range slur symbols.

Extensive slur size variations have not been implemented for several reasons:

- The lack of dimension registers (256 available are nearly exhausted in LATEX+MusicTEX) to record the initial sizes (horizontal and vertical) of this symbol *for each slur/tie* in order to make adequate links over glue and to close it with the symmetrical symbol.
- We do not think it wise to introduce in MusicTEX itself a great number of macros which would be little used by most users and would overload the restricted TEX memory, resulting in too many `TeX capacity exceeded` crashes.

The drawback of complexity. Due to the large amount of information to be provided for the typesetting process, coding MusicTEX sometimes appears to be awfully complicated to beginners, just as does the real keyboard or orchestral music. This is a necessary inconvenience to achieve its power and we can only encourage people just wanting to typeset a single voice tune to ask their local TEX guru for a set of simplifying macros...

Some examples

Many examples can be typeset from the MusicTEX distribution, but they are not included here for the sake of brevity. However we chose to produce a small type size version of

1. *Le cantique de Jean Racine* by Gabriel Fauré, in a transcription fit for organ accompanying.
2. The *Ave Maria* originally called *Méditation* by Charles Gounod, in a transcription fit for organ and voice or violin (the original is written for both a piano and an organ, which are difficult to find in the same room).
3. A part of a personal composition for the piano heavily using beams.
4. The beginning of Joseph Haydn's *aria* from the Creation, transcribed for organ and voice.

Bibliography

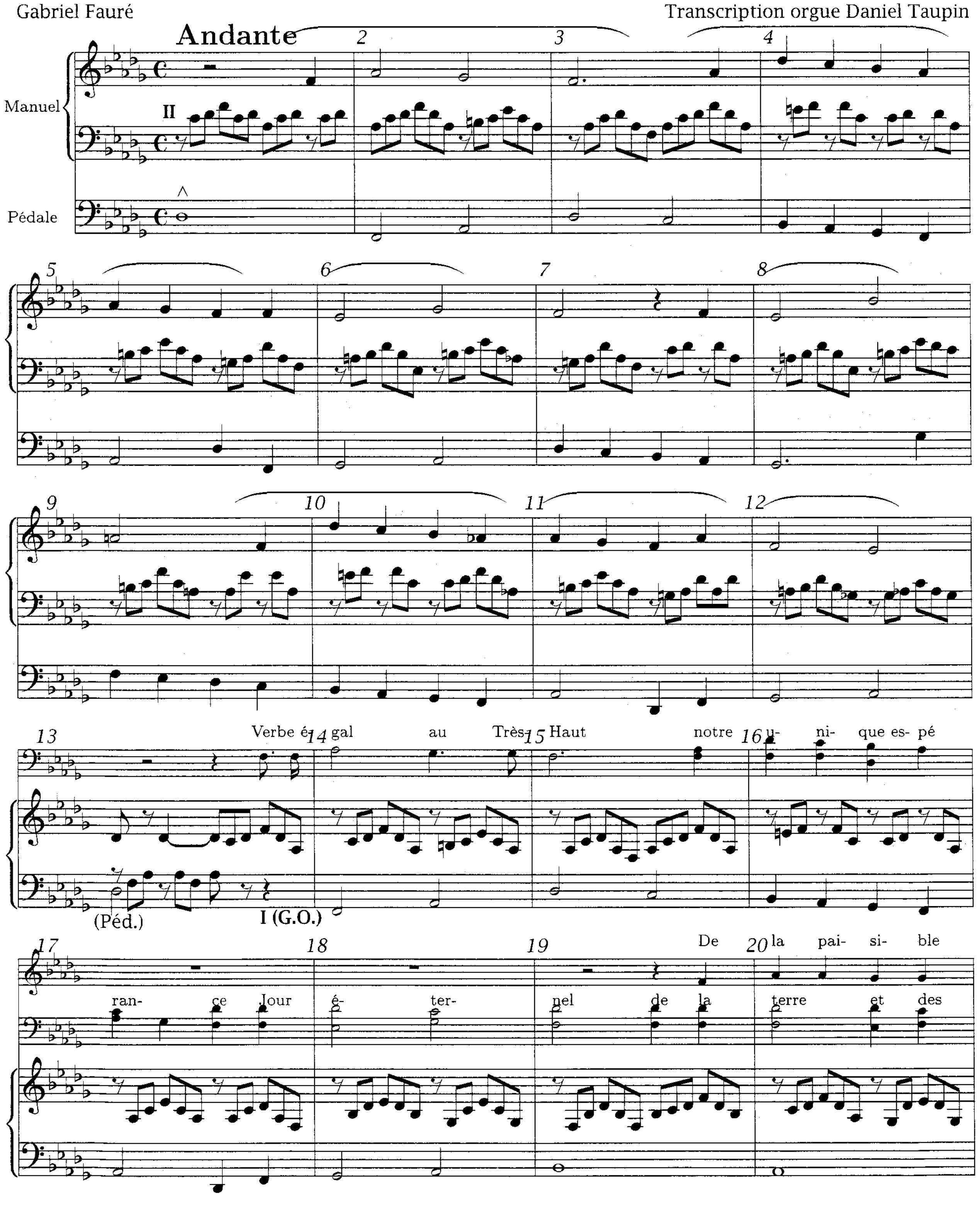
Andrea Steinbach and Angelika Schofer, *Theses* (1987, 1988), Rheinische Friedrich-Wilhelms Universität, Bonn, Germany.

Hans Kuykens, *MIDI2TeX* (1991), available at anonymous ftp: `obelix.icce.rug.nl` (in directory `pub/erikjan/MIDI2TeX`).

Cantique de Jean Racine

Gabriel Fauré

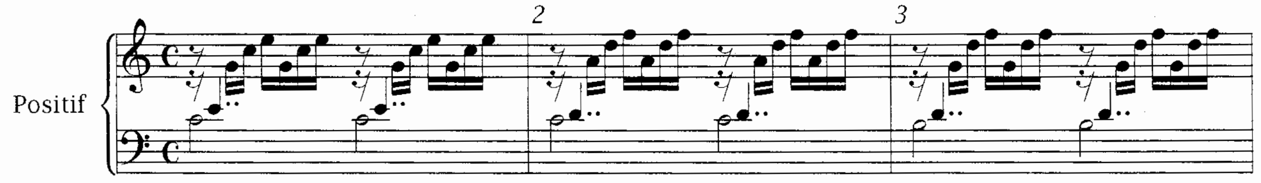
Transcription orgue Daniel Taupin

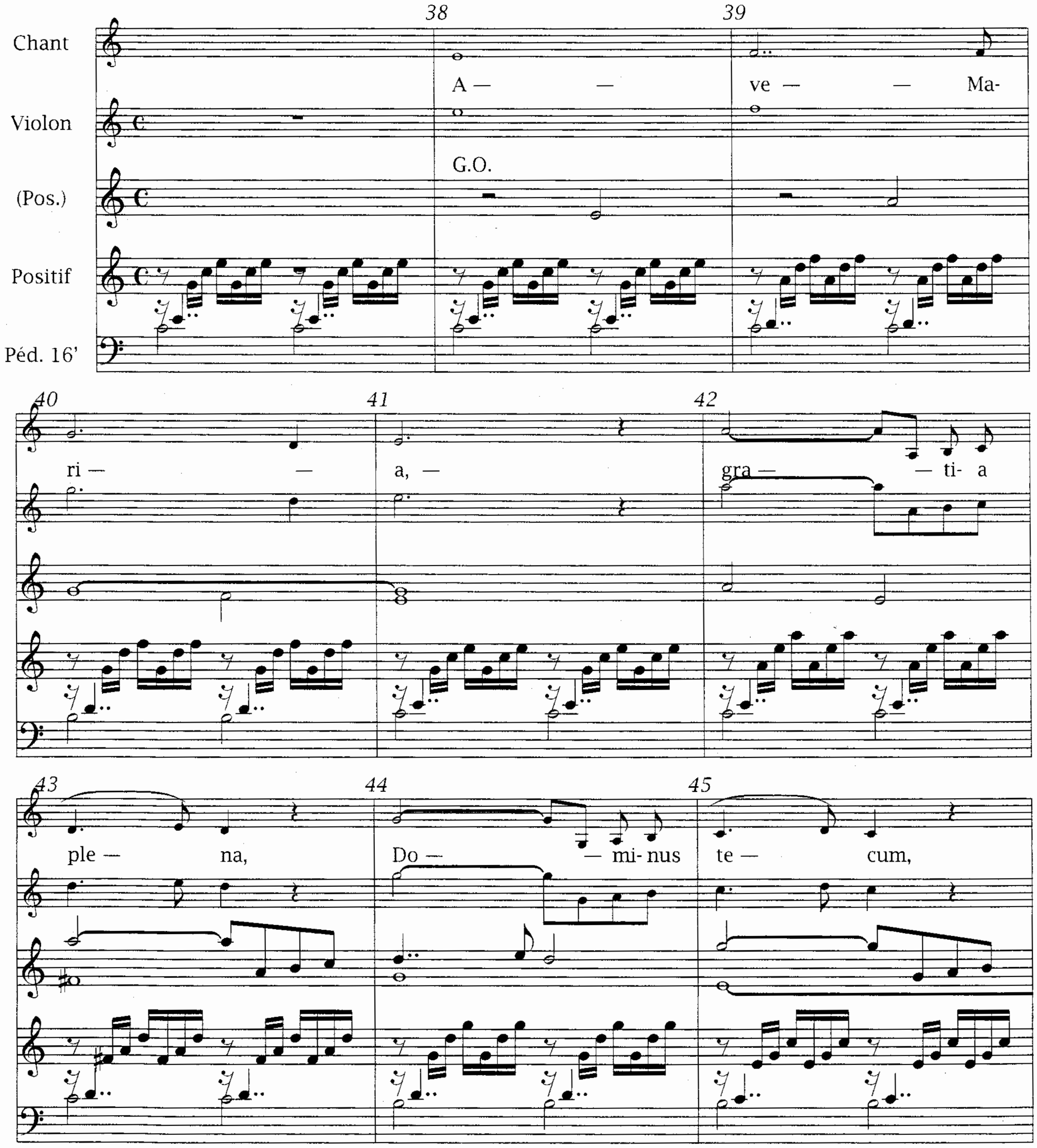

Andante
Manuel
Pédale
II
(Péd.)
I (G.O.)
Verbe é- gal au Très- Haut notre u- ni- que es- pé
ran- ce Jour é- ter- nel de la terre et des
De la pai- si- ble

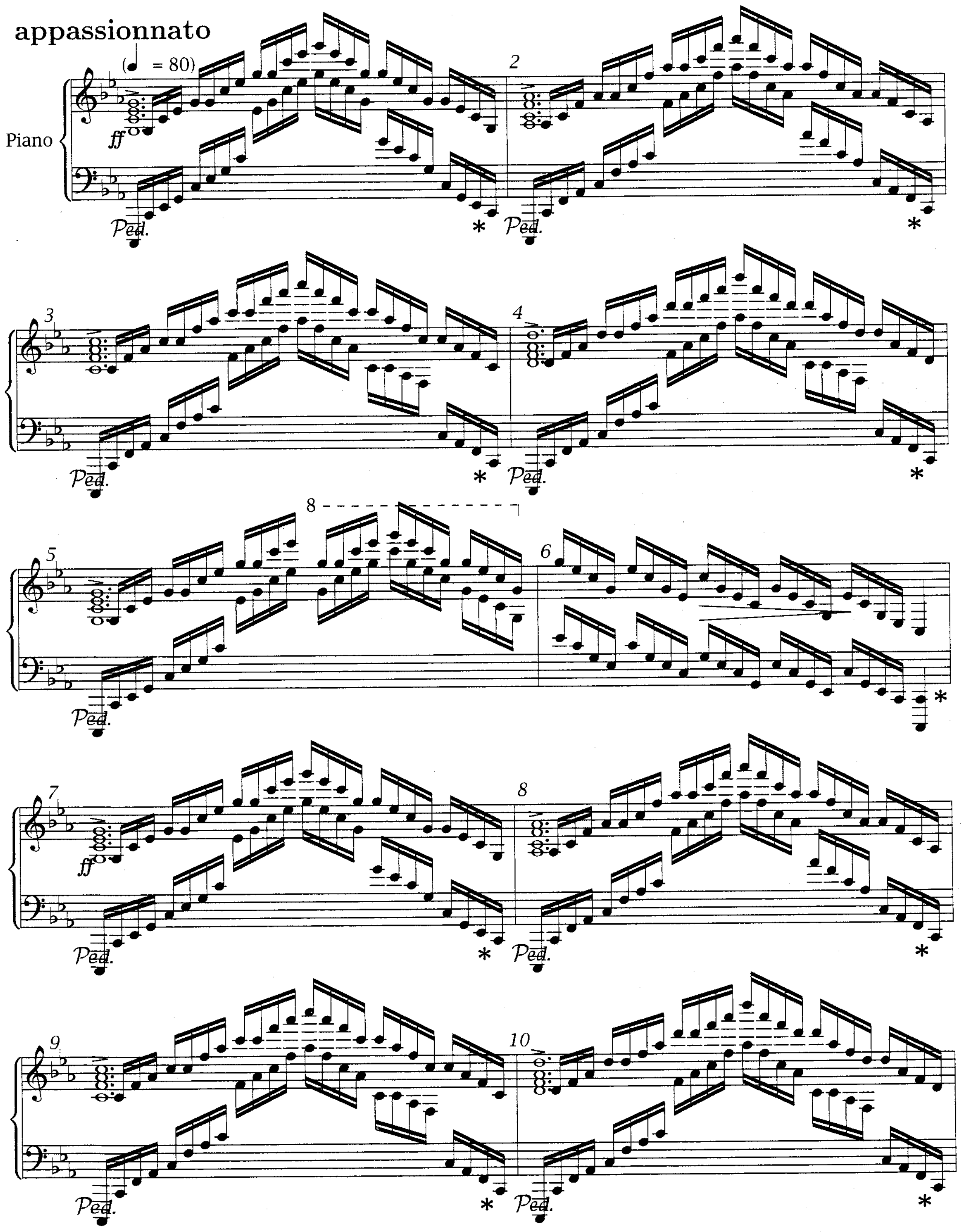
Méditation – Ave Maria

Charles Gounod & J.-S. Bach

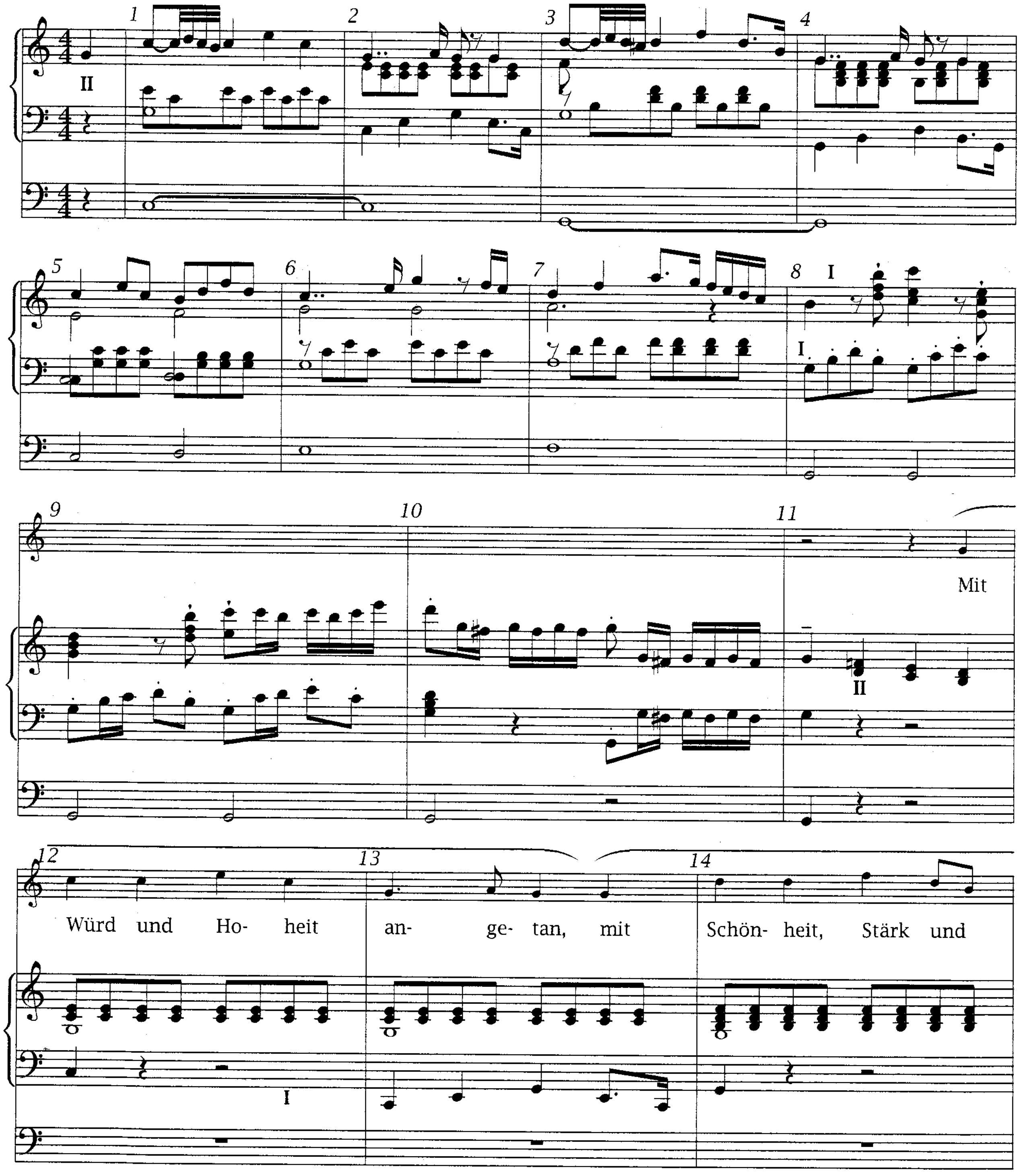
Transcription orgue+soliste Daniel Taupin & Markus Veittes






appassionnato
(♩ = 80)
Piano
ff
Ped.
*
2
3
4
5
6
7
8
9
10

Aria No. 24
(The Creation)
Joseph HAYDN
Transcription for organ and tenor, D. TAUPIN (1990)



A Format Compilation Framework for European Languages

Laurent Siebenmann
Mathématique, Bât. 425
Université de Paris-Sud
91405-Orsay, France
Internet: `lcs@matups.matups.fr`

Abstract

In the third and final version of TEX (fall 1989), Donald Knuth provided his TEX with facilities for multilingual typography: fonts of 256 characters, hyphenation for many languages, and virtual fonts — to mention three of many wonderful new features. However, an arbitrary limitation of TEX forces one to load all language hyphenation patterns with naked INITEX, a primitive and notoriously unfriendly version of TEX that has hitherto been mostly reserved for TEX wizards. This article presents FORMAT-DUMPER, a TEX macro package in the form of a directory of inter-related `.tex` files; it offers a pleasant interactive environment for the creation of multilingual formats and should thus enable ordinary TEX users to build precompiled formats with a cocktail of language and other features appropriate to their needs. FORMAT-DUMPER was originally posted in April 1992 in a bilingual French-English version that nevertheless already served the major TEX format cores: Plain, LATEX, AMS-TEX, and LAMS-TEX. Along with some current features, this article mentions numerous possible improvements. To allow FORMAT-DUMPER to be fully multilingual within the realm of European languages that use a basically Latin alphabet, I propose use or adaptation of Johannes Braams' `babel`, assimilating its style-independent features. Thus FORMAT-DUMPER should become a useful adjunct to an updated `babel`. With this in view, the article concludes with a carefully motivated discussion of the stellar language switching mechanism of Babel using however modified nomenclature based on the two-letter language codes recently introduced by Haralambous.

Introduction

Since the development of the TEX kernel was terminated by Knuth in 1989, a number of important problems have come into focus that must now be solved within the somewhat arbitrary constraints of TEX version 3. This article discusses one of them, the compilation of non-English or multilingual formats — restricting attention to the problems raised by European languages that use a (possibly accented) Latin alphabet.

Version 3 of TEX forbade introduction of new hyphenation patterns during the normal operation of the program TEX. More precisely, only the special setup of TEX called INITEX is henceforth able to interpret the command `\patterns`. What is more, although Knuth's documentation gives no warning, even INITEX is unable to interpret `\patterns` in case a precompiled format has been loaded by INITEX.

It seems to me to be a perfectly reasonable and worthy challenge to modify TEX so as to obviate this limitation. However, I suspect that no such modification of TEX will be in wide use before the next century dawns. We should therefore endeavor to live comfortably with TEX as Knuth left it.

[*A note for TEXperts*: One can of course wonder whether this is a documentation bug, a program bug, or a widespread implementation bug. I believe it is an accidental documentation bug and an intentional program limitation. As anecdotal evidence I point out that, on page 453 in *The TEXbook* it is asserted that "... the pattern dictionary is static: To change TEX's current set of hyphenation patterns, you must give an entirely new set ...". In fact, this statement is a hangover from earlier versions. Indeed, since version 3 appeared, `\patterns` (while available) behaves cumulatively as `\hyphenation` always has, and Knuth says as much in his *TUGboat* article (Knuth, 1989) introducing version 3 of TEX. Bernd Raichle `<raichle@informatik.uni-stuttgart.de>` confirmed on the basis of his reading of *TEX The Program* that the limitation lies in the program. Further, Raichle points out another mild inaccuracy in the documentation of `\patterns`: The statement in Appendix H of *The TEXbook*: "All `\patterns` for all languages must be given before a paragraph is typeset" is not strictly accurate. Only a *second* linebreak pass (that for hyphenation) invalidates `\patterns`. Indeed, like `\dump`, it causes the hyphenation trie to be packed; and a second pass can be inhibited, if you set `\pretolerance` to a high value.]

The upshot of this is that the *precompiled* versions of standard formats such as Plain, LATEX, AMS-TEX and LAMS-TEX that one encounters *cannot* be enhanced to handle a language whose hyphenation patterns have not been installed at the time of the original format compilation. Under TEX in version 2 there was the possibility of flushing out existing patterns at any moment and replacing them with new; these patterns could be arranged to serve two languages at once using a trick that Knuth attributes to J. Hobby (see early versions of *The TEXbook*).

In practice, this presently means that the average TEX user has available only one or two languages chosen by the TEX guru who compiled the formats at the given TEX site. In continental Europe most sites support English and (hopefully!) the national language. Current TEX implementations usually have space for a third language, but — for lack of hyphenation pattern capacity (`trie_size`, and `trie_op_size`) — not more. There being usually no unanimous choice for a third language among a multitude of choices,[1] it is rarely present. But most European users would greatly appreciate having their TEX 'ready to go' in some other language or languages they know. This situation is a deplorable obstacle to the optimal use of TEX in languages other than English, and to wide dissemination of non-English documents in `.tex` format once they have been created.

There is another problem of which *all* scientists and scholars are at least subliminally aware. In a high proportion of scientific or scholarly bibliographies, some titles are in 'foreign' languages. The current custom of using tiny fonts for such bibliographies and inhibiting hyphenation is a barely tolerable stop-gap measure, and, for multicolumn styles, it fails badly, causing incorrect linebreaks. In the near future, authors will, I hope, make it a habit to specify the languages using conventional control sequences (Haralambous, 1992, 1993). Those who employ a multicolumn style have strong reason to demand the ability to *quickly* create a format supporting the required foreign languages. Publishers and others who care about the finer points of typography should *always* demand this ability. This need suggests that, in the near future, *big* TEX implementations should routinely be able to handle somewhat more than the half dozen or so languages they do currently with `trie_size`=32 kilobytes. It also reveals a need for just 'basic' services for *many* foreign languages.[2]

The Format-Dumper solution in outline. I feel the best way to offer appropriately designed multilingual formats with TEX 3 is to make the use of INITEX for format compilation an easy matter that any confirmed TEX user will happily undertake on his own. My view is not yet widely shared, probably because INITEX is reputedly an unfriendly animal of which ordinary users should steer clear. More widely accepted is the notion that just tomorrow we all will have computers of infinite capacity and infinite speed, equipped with precompiled universal formats containing all the language features we need. I am skeptical of this optimism and suggest that Knuth is another skeptic:

Suppose you were allowed to rewrite all the world's literature; should you try to put it all into the same format? I doubt it. I tend to think such unification is a dream that's not going to work.

TUGboat, vol 13 (1992), page 424.

I have constructed a provisional setup for exploiting INITEX, called FORMAT-DUMPER. It currently comes in two flavors "-cm" and "-ck", serving Computer Modern and Cork norm font encodings respectively. The master posting is on `matups.matups.fr`, and mirror postings are available on faster CTAN servers (see these proceedings).

On the basis of this experience, which is just beginning to branch out from its bilingual French-English core, I hope to be able to suggest here features for a general framework that should apply to all European languages. It is surely too early to

1 In Haralambous (1992), where two-letter language codes were proposed, the existence of hyphenation pattern files for the following is asserted: Croatian HR, Czech CS, Danish DA, Dutch NL, Finnish FI, French FR, German DE, Hungarian HU, Italian IT, Norwegian NO, Portuguese PT, Romanian RO, Slovenian SL, Spanish SP, Swedish SW, UK English UK, US English US, all these supported by `babel` (Braams, 1991); and moreover Catalan CA, Estonian ES, Icelandic IS, Lithuanian LT, Polish PL and Slovak SK.

2 Titles include a variety of punctuation, so punctuation is clearly 'basic'.

pick best possible choices, but it is high time to evoke possibilities before a critical public.

I discuss FORMAT-DUMPER under the headings:

- exploitation of standard format files rather than *ad hoc* versions modified for the needs of European languages;
- assemblage of diverse tools in a directory rather than in a monolithic file;
- an interface based on programmed 'dialogs' between INITEX and the user; and
- a stellar protocol for switching language conveniently in multilingual formats.

Interspersed among these are numerous comments on problems raised by the use of active characters in multilingual formats.

Even if FORMAT-DUMPER develops little or ultimately disappears, I expect that some of the new features I discuss will be adopted in fully developed multilingual frameworks of the future.

Format-Dumper plus Babel. FORMAT-DUMPER is, I believe, the first system to address the problem of systematizing format building under INITEX. On the other hand, the first system for organizing multilingual TEX is babel by Johannes Braams (Braams, 1991), a system one finds posted on many servers. It provides adaptations of LATEX and Plain TEX to an impressive list of European languages.

What is the relation of FORMAT-DUMPER to babel? In its bilingual form FORMAT-DUMPER was independent of babel because the French language features were developed by Desarmenien and others before babel was conceived. Much of the effort in the bilingual version of FORMAT-DUMPER went to serving French users of AMS-TEX and LAMS-TEX who were not served by babel. FORMAT-DUMPER offers no services that are specific to LATEX or to any format — although there are a good many patches for the major formats (notably AMS-TEX and LAMS-TEX) in order to permit character activation. Its emphasis is on low-level things: the installation of patterns, the character activation just mentioned, font and accent administration (this part called Caesar).

babel gives little help in managing INITEX, perhaps because babel was designed before the new strengths and limitations of TEX 3 were fully assimilated. On the other hand, after four years of development, babel covers an impressive spread of languages, while FORMAT-DUMPER is just a prototype that is clearly incomplete. babel's emphasis has been on serving LATEX.

From the users' point of view in 1993, the sum of both and much more are wanted, since so much remains to be done for multilingual TEX.

Unfortunately, in a fully multilingual context, this sum cannot remain disjoint. Certainly FORMAT-DUMPER must exploit the vast compilation of macros for the typography of many nations that babel has built up. Further, the examination of language switching that I make in the closing section of this article leads me to conclude that babel's stellar language switching mechanism is effective, natural, and even capable of extension for worldwide use. Indeed, I see no reasonable alternative to it. Consequently, to become truly bilingual, FORMAT-DUMPER must in some manner join forces with babel.

I believe most of the existing and proposed features of FORMAT-DUMPER could be happily married with most low-level features of babel. This seems an attractive way to progress rapidly; in practice this means that the features of babel not specific to LATEX should be physically separated for ready use by FORMAT-DUMPER. Perhaps the LATEX features will be absorbed into version 3 of LATEX, (cf. Haralambous, 1993).

It is still quite unclear to me whether FORMAT-DUMPER should dissolve into the babel system or whether it should continue an independent life by interfacing with a future version of babel.

Contributions from Haralambous, 1992. In a 1992 article that indicates directions for multilingual developments centered on Norbert Schwartz' DC fonts, Y. Haralambous proposed a succinct language switching syntax \FR, \DE, \UK, \US, ... that already has an intriguing history. It is worth quoting the relevant passage:

"The DC fonts and TEX 3.xx's language switching features require new macros, which will also have to be standardized.

"These macros are of two kinds:

" 1) macros for accessing accented or special characters which are not available in the Computer Modern fonts [... omitted]

" 2) macros for language switching (see 'Language-switching macros', below). Since it is now possible to typeset a multilingual text where each language uses its own hyphenation rules, its own fonts and eventually its own direction of script, there must be a standard (and simple) way to switch between these languages.

"So if, for example, you want to include German or French words in your English text — 'Wagner's *Götterdämmerung* was very appreciated by the 19th century's *bourgeoisie*' — they will have to be hyphenated according to German and French rules. To indicate this to TEX, macros have to be selected which are easy to remember, short, and universally acceptable.

"For this, the standard has basically been taken from the standard 2-letter ISO 639 language codes as control sequence names [footnote on exceptions omitted] (see 'Language-switching macros', [omitted]).

"The previous example is then written:

```
\US Wagner's
\DE {\it G\"otterd\"ammerung\/}
\US was very appreciated
by the 19th century's
\FR {\it bourgeoisie}
```

I considered these recommendations very positive and felt that they deserved to guide the development of language switching in FORMAT-DUMPER. When presented orally at the EuroTEX conference in Prague (September 1992), they raised mild objections from persons who felt that more explicit names such as `\french`, `\german`, `\ukenglish`, `\usenglish`, ... would be more suitable. On the other hand, the idea of using an ISO standard for these new lower-level macros had the great advantage of leaving undisturbed existing higher-level macros like `\french` and `\german`. The use of capitals (`\FR` not `\fr`) further reduces the probability of conflict with existing macros.

Recall that in becoming truly multilingual FORMAT-DUMPER is faced with the awkward matter of assimilating the low level language features from the present babel that constitute a core of essential services which all users will want, while leaving aside a plethora of optional high level features that belong more to a style file than to a base format. I therefore set out to implement Haralambous' syntax in a simplest possible fashion. However, I quickly found myself skating on thin ice; the difficulties described in the closing section made me fall back on the approach of babel.

In retrospect, one can perceive warning signs even in the above quotation. Why was the syntax not as follows?

```
\US Wagner's
{\DE\it G\"otterd\"ammerung\/}
was very appreciated
by the 19th century's
{\FR\it bourgeoisie}
```

This syntax is more natural and and reduces the uses of \US from two to one. For this to work, \US, \FR and \DE must respect TEX grouping.

I surmise that Haralambous intended at the time of publication (Haralambous, 1992), that \US, etc., cause *global* changes which do *not* respect TEX's grouping. This turns out to be rather objectionable, because such global changes are confusing to the user in the presence of pre-existing language macros that do respect grouping, beginning with those of Desarmenien for French in the 1980's. Furthermore, Knuth warns (*The TEXbook*, p. 301) that use of *both* local and global changes of the same entity can lead to a slow poisoning of TEX's grouping mechanism.[3]

But this answer raises a second question. Why would Haralambous have proposed `\global` changes? There are known approaches to implementing language changes by using grouping (and respecting grouping); they do work but they encounter difficulties that I will explain below in motivating the stellar protocol.

When the French translation (Haralambous, 1993) appeared in spring 1993, the macro syntax proposal was omitted without comment. One unfortunate consequence of this disappearance of \US, \FR, \DE, etc. is that the related internal macros I will propose for language switching may not be welcomed. On the other hand I would be very pleased to contribute to rapid rehabilitation of these handy control sequences in a form respecting grouping.

The reconstruction of babel's language switching in the closing section offers somewhat more than modernized notation. I feel that use of a virtual language EC at the center of the star in place of US is a step in the right direction — the 'communal' organization functions slightly better than the 'parental' one. It also suggests a tree-like structure for a worldwide system.

Use of Standard Formats

The use of *unmodified* standard versions of the `.tex` files defining standard formats is recognized to be a feature vital for the stability, compactness and clarity of any TEX setup. This discipline is accepted by FORMAT-DUMPER; it does *not* mean that undesirable

3 At one point there was also a failure of babel's system to respect grouping; see the \gdef on page 294 of (Braams, 1991).

behavior of a format cannot be modified; but rather that modifications will all be done by 'patch' files external to the standard format files.

By dint of somewhat 'dirty' patching trickery it becomes possible to use standard `plain.tex` even when Cork norm fonts are used. The read-only memories (=ROMs) that provide the low-level routines for micro-computers usually provide a protocol for patching; similarly it would be a good thing if future versions of basic formats were built to make patching less tricky.

Just a few kilobytes specific to FORMAT-DUMPER are needed to compile a typical individual's desired format. Thus it is possible, for example, for a student in Australia with a microcomputer and a modem of speed only 2400 baud to obtain, by `ftp` from a European site, enough of FORMAT-DUMPER in just a few minutes to automatically compile a good French-English bilingual format with the help of standard formats he already possesses.

Assemblage of Diverse Tools as a Directory

Format files, language hyphenation files, patch files, language files for typography and typing, and, finally, dumper files organizing the foregoing — these are the main constituents of the FORMAT-DUMPER directory. There is even a catch-all `initexme.tex` dumper file which leads the user to all possible format choices.

The patch files for French required to accommodate possibly active punctuation are non-trivial (although compact) with 1993 $\mathcal{A}$$\mathcal{M}$$\mathcal{S}$-TeX, and LAMS-TeX. I expect that optimal use of many other languages will require some character activation, but hopefully less than for French.

As FORMAT-DUMPER covers more languages and formats, the number and total volume of files in the FORMAT-DUMPER directory will become considerable. On the other hand a single user wanting to build a specific format needs only a few kilobytes of files. How can one enable the user exploiting ftp to get just what he needs? One interesting possibility (among many) is to have a one-file 'scout' version of FORMAT-DUMPER which produces not a format but rather a 'roster' file, each line of which is of the form

`mget` ⟨*filename*⟩

This roster will serve under most operating systems to fetch automatically by ftp all the required files, and not more.

The thorny issue of character activation. Without this challenge, I might never have bothered to create FORMAT-DUMPER! However, only the impact on the design of FORMAT-DUMPER is of interest here.

Some TeXperts (myself included) feel that activation of several characters, notably `;:?!` (which in French typography should yield subtle preceding spacing) is the only way to make French typing for TeX at once convenient, portable, and typographically correct. French is perhaps the most troublesome language in this regard, but others have similar problems; for example, German TeX formats often activate `"` for a variety of reasons. Such activation unfortunately requires that formats and macro packages be 'cleaned up'; FORMAT-DUMPER does the necessary patching non-invasively. Hopefully, someday soon, the standard versions of major formats will be 'clean' in this sense.

Other TeXperts, firmly believe that category change is too troublesome to cope with — arguing that there will always be macro packages that have not been 'cleaned up'. They accept less convenient typing, for example `\;` for semicolon in prose, or they have to use a preprocessor. For them, an option keeping `;:?!` etc., *inactive* (i.e., of category 12 = other) will be provided by FORMAT-DUMPER.

There is a residual category problem to be investigated in multilingual TeXs. It is known to be troublesome to switch category codes at times when TeX is reading ahead during its macro expansion process, for instance when it comes to expand the middle of a big macro argument. This arises because category code change cannot influence what has already passed TeX's lips. If the act of changing category code is dangerous, then we should perhaps not change category code in the process of changing language! That has indeed always been a policy of FORMAT-DUMPER.

Then what category choice is appropriate for an English-French-German format? There is a practical answer in this known case: have `;:?!` active and `"` inactive, but allow users of German to activate `"` *with due care* when they wish — using a command `\flexcat!\activate"`. An error warning is issued by `\flexcat!` in the rare cases when category switch is dangerous at that point.[4] A language change to French there could cause bad punctuation (*n'est-ce pas ?*) — if one were to change category

4 Such dangerous points cannot be predicted on general principles. But, it may be helpful to note that they are more or less the points where 'verbatim' macros break down.

code along with language. A 'danger warning' macro for category change is perhaps a new idea; here is a quick explanation. `\flexcat` temporarily changes the category of `!` to 11 (letter) and then examines the category of the following `!`; if it is not 11 an alarm goes off. For more details see my July 1993 posting mentioning `\flexcat` in the GUT TEX forum archived on `ftp.univ-rennes1.fr`.

I am therefore optimistic that category activation and multilingual TEX formats can be mixed if due care is taken. There always remains an 'all inactive' option, but it would be sad to see TEX become less friendly in an international context!

A User Interface Based on Programmed 'Dialogs'

This means dialogs between INITEX and the user orchestrated by a 'dumper' script. Mike Downes has written a very valuable guide for building such dialogs (Downes, 1991).

This interface makes the FORMAT-DUMPER system nearly self-documenting and an incredible number of distinct compiled formats becomes readily available to a diverse population of users. FORMAT-DUMPER is a self-serve 'format supermarket'!

The user will be able to select languages at will from a list of languages currently supported, subject only to limitations of TEX capacity. For each language the user will specify (from a short list of options) which characters he will make active. One of the options should always be *none* (i.e., no new active characters)! The format being produced will, for reasons explained above, have as active characters all those characters designated as active by the user for *one or more* of the languages included. Long experience in having `;:?!` active for English in a French-English format encourages me to believe that this *modus vivendi* will be acceptable.

There are many options it is wise to include into a format dumped by FORMAT-DUMPER rather than `\input` them repeatedly later. The $\mathcal{A}\mathcal{M}\mathcal{S}$ math fonts are one of my steady personal choices. It will be possible and desirable to put more and more good things into a compiled format, in order to to fit the users special needs.

To prevent this wealth from bewildering the user (and those who have to handle his or her `.tex` files at a later date) devices are needed to recall the salient features of the format built and permit an equivalent format to be created in any other environment. Even the casual user will want this under MSDOS operating systems, where the format name is limited to $8+3$ letters and hence inadequate as a source of information.

Here are two devices for format identification. By using the `\everyjob` primitive, a telegraphic summary of the format can be placed on the log at each launch. Further information should be provided by an optional command `\ShowFormatInfo`. This should indicate all the essential choices made in creating the format. This information should be formatted as `.tex` comments suitable to become part of the header of any `.tex` file. This should assure world-wide portability of the `.tex` files for any format produced.

Format compilation, for a book say, is a process that I tend to repeat often until just the right recipe is found: just the right fonts and macros, no more. There will be a way to have FORMAT-DUMPER *learn* to anticipate what one wants, so that most multiple choices can be bypassed as the default choice, i.e., by simply striking the return key. Here is the idea: FORMAT-DUMPER should learn to make the right choice the 'default' choice. For this, one can have FORMAT-DUMPER make the default choice be the choice made on the previous dumper run; choices would therefore be written to an auxiliary file. (If the last run is not recent, as measured by `\today`, one might reasonably revert to 'default defaults'.)

Using compiled formats to good effect. At a later stage, it saves much time to build upon formats produced by FORMAT-DUMPER, say `fe-LaTeX`, to obtain a more temporary format that attacks your typesetting *almost instantly*. If your (LA)TEX often spends many seconds chewing through macros before starting genuine typesetting, you have some tricks to learn.

They will be illustrated by an example. Given a `.tex` file for (say) `fe-LaTeX`, give the name `x.tex` to the segment of up to (and possibly including) `\begin{document}`, and let `y.tex` be the rest. Then add `\dump` to the end of `x.tex` and compile with INITEX, normally using the syntax

```
initex  &fe-LaTeX  x
```
(1)

A format is dumped; let it be called `x-LaTeX`. This new format is the one that then attacks `y.tex` instantly, using VIRTEX with the syntax

```
virtex  &x-LaTeX  y
```
(2)

On a unix system, one can alias this ephemeral but oft repeated command as (say) `ty`; further one might use 'piping' to make `ty` also preview the result.

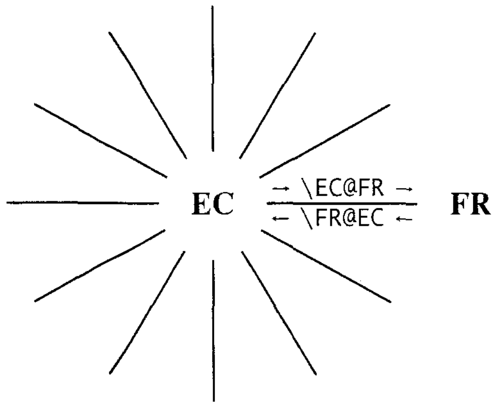
Regrettably, a number of routine but system dependent technicalities may necessitate help from your system manager. The most frustrating is perhaps difficulty in accessing a freshly dumped format. I recommend wider user education concerning the way TEX accesses the various files it needs.

It is a praiseworthy initiative of *Textures* on Macintosh computers to incorporate into VIRTEX all the functionality of INITEX. This means that *Textures* users are less intimidated by format dumping. Instead of the command (1), they have *Textures* compose `x.tex` using format `fe-LaTeX`; and instead of (2), they have *Textures* compose `y.tex` using `x-LaTeX`.[5]

Switching Language

The stellar protocol described here is morally that of babel, but I have permitted myself liberal changes of detail in the hope of clarifying ideas and influencing future versions of babel.

The proposed center of the star is not English but rather a virtual language whose two-letter tag should be EC (for European Community); it would have neither hyphenation patterns nor exceptions. (More details on making EC a convenient institution come later.)



For each language, say French, a number of language-change control sequences should be defined.

(a) \FR for a middle-level language change, as proposed by Haralambous (1992).
(b) \EC@FR and \FR@EC, macros which are of a lower level than (a).
(c) \l@FR is TEX hyphenation system number for FR. \language=\l@FR switches to it. This is the lowest level.[6]

The shape of this protocol is *stellar* since all languages are explicitly related to the virtual central language EC as the figure above indicates and *not* directly among themselves.

To motivate this design we now examine the nature of the switching problem and the weaknesses of simpler designs.

Unlimited complexity of language. The macro \FR must be more complicated than one first thinks because its action in general should *depend on the current language at the time it is called.* Indeed, special features for the current language have to be neutralized. The features I have in mind are chiefly the behavior of punctuation and spacing that are peculiar to the current language, but they may include typing conventions and there are many additional bits and pieces. (The hyphenation patterns and exceptions themselves do not pose any problem, since by calling new ones we automatically put aside the old.)

These bits and pieces are sufficiently important that they cannot be ignored, and sufficiently numerous that one does not have time to reset all the features and parameters that some other unspecified language might conceivably have altered. There is an attempt in Haralambous (1992) to put an *a priori* bound on the things that that will change under language switching; I feel that this is impractical, and fortunately we will find it unnecessary. Let us look at some bits and pieces.

The Czechs have a rare but logical and unambiguous convention for splitting hyphenated words at line ends: the hyphen is programmed to survive at the beginning of the following line. There is the question whether the parts of a word spelled

5 There was an unfortunate bug in versions 1.4 and 1.5 of *Textures* that virtually forced one to quit *Textures* before using a (re)compiled format. The long life of this bug, now fixed by version 1.6, confirms my impression that even in the exceptionally good *Textures* environment, format compilation has been underused!

6 (a) babel's `\selectlanguage {francais}` corresponds roughly to \FR. In fact the syntax `\selectlanguage {FR}` should be retained in programming.
(b) babel's `\extrasfrancais` corresponds roughly to \EC@FR, and `\noextrasfrancais` to \FR@EC.
(c) babel's `\l@francais` corresponds to \l@FR.

with a hyphen should be subject to hyphenation at linebreaks. French spacing around punctuation involves `\frenchspacing` which *suppresses* extra spacing after punctuation (conventional in English) and *adds* peculiar extra spacing *before* the four punctuation points `;:?!`. Another detail is the positive `\lccode` the French assign to the apostrophe (English gives it `\lccode=0`). I expect the list of such language dependent features will never be complete.

Accent style becomes a nightmare for TEX. With CM fonts, French users usually prefer to suppress accents on capital letters, although with a suitably designed font family, French typography recommends their use. Incidentally, suppression can be gracefully handled by adding a few hundred octets of macros to the accent administration package 'Caesar' currently used by FORMAT-DUMPER.

Far more vexing for multilingual format building are disagreements in accent positioning between, for example, French and Czech (Zlatuška, 1991); thus I permit a digression here. At present this disagreement seems to require that different font systems be used for French and Czech, at considerable cost — simply by reason of a small number of misplaced accents. Although this is wasteful, let it be conceded that the setup can be gracefully handled by the NFSS (New Font Selection System) of Mittelbach and Schöpf. Indeed, 'language' can be one more 'property' analogous to 'shape' or 'size'. This solid but weighty solution is recommended by Haralambous (1993) and is materially supported by NFSS version 2.

I mention that there may perhaps be an efficient solution based on the stellar language switching we are studying, and the notion of a SPECIAL within a character description in a virtual font.[7] Consider the task of enhancing a future virtual version of the DC fonts of N. Schwartz to allow the positioning of accents favored by Czech typography (Zlatuška, 1991). Each language change into or out of Czech should be marked in the `.dvi` file by a TEX `\special` command. For implicit language switches induced by TEX's grouping, the `\aftergroup` primitive helps to insert the `\special` command. Nevertheless, the macros `\CS@EC` and `\EC@CS` for Czech are the only macros in the switching scheme that need to be enhanced. The virtual DC fonts would then be enhanced by including SPECIALs within the MAP description of each accented character altered for Czech, indicating the modified accent positioning. This would, in turn, require that drivers learn to interpret these specials in the intended order. (These SPECIALs and their modifications are ignored by current drivers.) By this method, single virtual font can become a sheaf of virtual fonts indexed by a parameter or parameters. In this context, the parameters correspond to language related variations, and a language change typically causes tiny alterations on a few font characters. This solution is a 'pipe dream' — but hopefully one with real potential.[8] (I believe it worthwhile to dream up SPECIAL extensions of virtual font mechanisms, as that will, in the long run, stimulate the improvement of drivers and driver standards.)

I have argued (with digressions) that there is a limitless number of things `\FR` would have to do if it were not known what language we are switching out of. Presently, I will observe that it is easy to have TEX keep track of the current language and that an efficient way to switch fonts is obtained by going between any two languages *via* the central language EC using the macros (b). But, before plunging into more detail, we really must be convinced that simpler pre-existing schemes are inadequate.

Language switching based on grouping. Many readers will surely have thought of a simpler language switching scheme based on grouping. Suppose we begin each typescript always in a preferred 'base' language, say US (where TEX began). Then, if we define `\FR` to switch from the base language to FR and similarly for `\DA` and `\CS`, the syntax

{\FR ⟨*French text*⟩}
{\DA ⟨*Danish text*⟩}
{\CS ⟨*Czech text*⟩}

will provide correct language switching. Alternatively, the LATEX 'environment' syntax:

\begin{FR}⟨*French text*⟩\end{FR}
\begin{DA}⟨*Danish text*⟩\end{DA}
\begin{CS}⟨*Czech text*⟩\end{CS}

could accomplish the same. In essence, why not rely on the powerful grouping mechanism of TEX to get us back to the base language at the end of a group, thereby economizing half the macros proposed in (c)? My answer is that this well-known approach is indeed valid but often *inadequate*!

7 There is another approach via an enhancement of the `\charsubdef` addition to TEX by Mike Ferguson `<mike@inrs-telecom.uquebec.ca>`, but Mike warns me it is not easy to implement.

8 Character shapes can change arbitrarily; the main restriction is that the font metric data cannot change.

It is inadequate when the current language has been declared at a higher grouping level than at the spot where we propose to switch language; indeed closing groups before switching — in order to return to the base language — is not an option there, since it has, in general, disastrous side effects. (In addition, one would not even know how many groups to close!) For example, the desirable syntax

```
\FR ⟨French text⟩
\begin{anyenvironment}
⟨French text⟩
{\DA⟨Danish text⟩}
⟨French text⟩
\end{anyenvironment}
⟨French text⟩
```

would be invalid, and clumsy to fix. Such clumsiness is unacceptable in many applications, such as literary criticism, history, art, travel, music, etc., where many quick language switches are required.

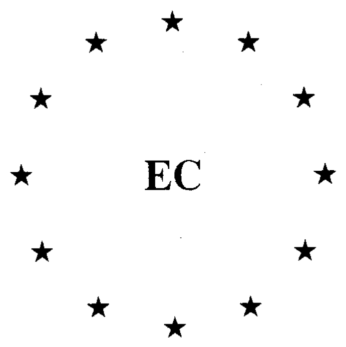
Note that the above example would pose no problem *if* the notion of language were encompassed by a fixed finite number of features such as hyphenation rules, fonts, and direction of script. Indeed it would suffice to have \FR, \DA, etc., each adjust all of these features; the change to Danish would *automatically* undo all the features of French. Unfortunately, this 'Cartesian' view of language seems inadequate, as was explained above.

Functioning of the stellar protocol. I hope the above analysis of alternatives will convince readers to take the stellar protocol seriously. Here are some details to explain how it provides valid language change, say in this last example.

At any moment, a reserved register \l@toks contains the tag of the current language, say FR, and there exists a macro \FR@EC pre-defined to cause reversion from FR conventions to the EC conventions. Its first duty is of course to change the value of \language from \l@FR to \l@EC. Its multiple other duties are to dismount all the special features that \EC@FR will have introduced at an earlier time.

The action of \DA is then double: First, it looks at the current language register \l@toks and uses what it finds (FR say) to revert to EC by \FR@EC. Second, it applies another pre-programmed macro \EC@DA to pass from EC to Danish, and replaces FR by DA in the current language register.

Thus, for each language (FR say), the crucial problem is to define two macros \EC@FR and \FR@EC to introduce and suppress national features.



Eurocentrism. The shape of this solution recalls the twelve-star circle on the European flag. It installs eurocentrism — since change from one national language to another passes through the formal center EC.

What should be the features of the artificial language EC? One wants it to be a sort of center of gravity to which one can easily revert. Thus, I would give it the consensus \lccodes used by Ferguson in his MLTEX. Null hyphenation patterns let EC serve in a pinch in lieu of a missing language, say \DA, at the cost of introducing soft hyphens by hand; \DA should then be \let equal to \EC, which prevents incorrect hyphenation for this language (and no more). Incidentally, \EC is defined like \DA, but of course \EC@EC does nothing.

For the rest, Knuth's features for English probably make a reasonable choice; then at least we all can remember what EC means!

Adaptability and extensibility of the switching scheme. For the purposes of FORMAT-DUMPER, the language change macros \FR, \US, etc., should initially be low to middle level. But, since various complex styles may be loaded later, there must be possibilities for enhancement of the features for any language.

Permanent enhancement is straightforward. On the other hand, a macro package might want to provide a macro \myfrenchextras to *conveniently add* French language features that will *go away* when we next leave French. This suggests a possible enhancement to the language change protocol described, which we now explain for the switch from French to Danish. \myfrenchextras should both introduce the extra features and add to a reserved token sequence \@ECtoks[9] an action \@undomyfrenchextras. Then, when we leave French by \DA, the enhanced mechanism would first

9 babel's \originalTeX corresponds roughly to \@ECtoks.

make \the\@ECtoks act, then empty \@ECtoks, and finally execute (as normally) \FR@EC\EC@DA. There are many possible variations to explore. Even the naive use of grouping will often produce adequate results.

One could allow more than one manifestation of a given language, say basic French FRb as presently supported by FORMAT-DUMPER, or classical French FRc as supported by french.sty of B. Gaulle (available on ftp.univ-rennes1.fr), or again FRx a French publisher's distinctive typography. A new manifestation of a language can clearly be added 'on the fly' by a user (without dumping using FORMAT-DUMPER), so long as the hyphenation scheme coincides with that of a manifestation already installed. It seems unlikely that rapid switching between such variants of French would be of any interest; hence any one could appropriate the standard macro \FR.

The opposite is true where dialects of a language are concerned, since conversations (say in a play) would require rapid switching. The 'closeness' of the dialects of one language could be most efficiently exploited using a treelike rather than stellar protocol (below).

Incidentally, to facilitate the above extensions, TEX should maintain a list in a standardized form of names of the installed hyphenation systems and another of the installed languages (with switching macros).[10]

A world-wide system. Naturally, one dreams of a world-wide system with the virtual language UN (for United Nations) at its center. Its natural shape would, I imagine, be treelike rather than stellar; EC might be one of several internal nodes corresponding to a language grouping. Two points of the tree are always joined by a unique shortest path (usually shorter than the one passing through UN) and this may permit a natural and efficient generalization of the stellar switching scheme that has been sketched above.

Such are the simple but powerful ideas underlying the language switching in babel. It seems clear that there are no insuperable obstacles to having future versions of babel and FORMAT-DUMPER collaborate — in order to provide convenient compilation of multilingual formats using INITEX.

I am pleased to thank Johannes Braams, Klaus Lagally, Frank Mittelbach, Bernd Raichle, and the referee for helpful comments on the preliminary version of this article.

10Beware that the correspondence of the two lists need not be exactly one-to-one. There may be different hyphenation systems for the same language (commonly minimal and maximal). And there may be more than one language manifestation/dialect using the same hyphenation system. In babel, an ephemeral external file language.dat bears this information at format compilation time; something more robust and accessible is called for — see (Taupin, 1993).

Bibliography

Braams, Johannes. "babel, a multilingual style-option system for use with LATEX's standard document styles". *TUGboat*, **12**(2), pp. 291-301, 1991; the babel system is available on the major TEX archives.

Downes, Michael. "Dialog with TEX". *TUGboat*, **12**(4), pp. 502-509, 1991.

Haralambous, Yannis. "TEX Conventions Concerning Languages". TEX and TUG News, **1**(4), pp. 3-10; this article with its useful tables is available in digital .tex form on ftp.uni-stuttgart.de.

Haralambous, Yannis, "TEX Conventions concernant les polices DC et les langues". Cahiers GUTenberg **15**, pp. 53-61, 1993.

Knuth, Donald. "The New Versions of TEX and METAFONT". *TUGboat*, **10**(3), pp. 325-328, 1989.

Taupin, Daniel. "Commentaires sur la portabilité de TEX". Cahiers GUTenberg, **15**, pp. 3-31, 1993.

Zlatuška, Jiří. "Automatic generation of virtual fonts with accented letters for TEX". Cahiers GUTenberg, **10-11**, pp. 57-68, 1991.

Bibliography Prettyprinting and Syntax Checking

Nelson H. F. Beebe
Center for Scientific Computing
Department of Mathematics
University of Utah
Salt Lake City, UT 84112
USA
Tel: +1 801 581 5254
FAX: +1 801 581 4148
Internet: `beebe@math.utah.edu`

Abstract

This paper describes three significant software tools for BIBTEX support. The first, `bibclean`, is a prettyprinter, syntax checker, and lexical analyzer for BIBTEX files. The second is `biblex`, a lexical analyzer capable of tokenizing a BIBTEX file. The third is `bibparse`, a parser that analyzes a lexical token stream from `bibclean` or `biblex`.

The current BIBTEX implementation (0.99) is based on a vague and ambiguous grammar; that situation *must* be remedied in the 1.0 version under development. Rigorous lexical analyzer and parser grammars are presented in literate programming style, and implemented as `biblex` and `bibparse` using modern software tools to automatically generate the programs from the grammars. `bibclean` also implements these grammars, but with hand-coded parsers that permit it to apply heuristics for better error detection and recovery.

Extensions of the current BIBTEX for comments, file inclusion, a `Periodical` bibliography entry, and ISSN and ISBN entry fields, are proposed and supported in these tools.

The impact of much larger character sets is treated, and grammatical limitations are introduced to ensure that an international portability nightmare does not accompany the move to these character sets.

`bibclean` is extensively customizable, with system-wide and user-specific initialization files, and run-time-definable patterns for checking BIBTEX value strings. A customized pattern-matching language is provided for this purpose. `bibclean` can also be compiled to use regular-expression patterns, or none at all.

All code is written in the C programming language, and has been tested for portability with more than 40 C and C++ compilers on several major operating systems. The distribution includes a large suite of torture tests to check new implementations. It is not necessary for installation to have the lexical analyzer and parser generator tools that process the grammars; their output code is included in the distribution.

The complete paper is too long for the TUG'93 Conference Proceedings issue; it will instead appear in the next issue of *TUGboat*.

A TEX User's Guide to ISO's Document Style Semantics and Specification Language (DSSSL)

Martin Bryan
The SGML Centre
29 Oldbury Orchard,
Gloucestershire, U.K.
Internet: `mtbryan@cheltenham-he.ac.uk`

Abstract

The importance of capturing structure information in documents that are likely to be re-used is increasingly being recognized. In both academic and commercial circles the role of ISO's Standard Generalized Markup Language (SGML) in capturing and controlling document structure is becoming more widely acknowledged.

While TEX itself does not utilize document structure information, many of its macro facilities, such as LATEX, provide some, albeit high-level, structure control. LATEX3 seeks to increase the level of structure recognition by adding recognition of attributes within macro calls to allow more than one interpretation of a structure-controlling markup tag.

ISO's approach to the problem of linking document structure to document formatting engines such as provided by TEX is to develop a language that can be used to add suitable sets of formatting properties to SGML-coded and other structured documents. The Document Style Semantics and Specification Language (DSSSL) has two main components: (1) a General Language Transformation Process (GLTP) that can take a document with a predefined input tree and transform that tree into the form required for subsequent processing, and (2) a set of Semantic Specific Processes (SSPs) that specify how specific operations, such as document formatting, shall be specified at the input of the formatter.

This paper explains these two processes, and shows how they can be used in conjunction with TEX.

Introduction

The advantages of adopting a structured approach to document markup have been known to TEX users for many years. Most of the macro languages developed for use with TEX use the names of the structural elements that make up a document to identify the way in which the document is to be presented to users. Notice that I did not use the word 'formatted' here. Today TEX is used to present documents to users on screen almost as often as it is used to prepare documents for printing, and this fact has to be a key factor in the development of any new language for describing how to present text. The characteristics of screens differ from those of paper, so the rules used for presenting information on screens have to differ from those used to present the same information on paper.

Within the Open Systems Interconnection (OSI) program of standards development at the International Organization for Standardization (ISO), there are two main standards for the creation and presentation of structured text. For data whose structure is controlled by the presentation process the Open (originally Office) Document Architecture (ODA) can be used to define the logical structure of typical office-related documents. For more general purpose applications, the Standard Generalized Markup Language (SGML) can be used to describe the logical structure of captured data, and the Standard Page Description Language (SPDL) can be used to describe the formatted result. ISO's Document Style Semantics and Specification Language (DSSSL), which acts as the link between these two forms, has been designed to allow systems to interchange information that can be used to convert logically structured SGML files into physically structured, presentable, SPDL files, or into the form required for processing by existing text formatters, such as TEX.

About DSSSL

As TeX users you will understand that there is a difference between the way in which information is presented on a page or screen and the way in which it is created and used. One of the key problems that has faced the SGML community is that the logical structure that is needed to guide users during data capture or data retrieval is not necessarily the best structure for text formatting. If users are constrained to a model that reflects the way in which the document is to be formatted they are likely to object to the need to capture data in a structured format (and, given human nature, are likely to go to the opposite extreme and insist on creating totally unstructured, "What You See Is All You'll Get" (WYSIAYG), documents).

What is needed is a coherent means of describing the relationship between elements used to navigate through an information set and the objects used to present specific parts of the information to users. The transformation that is required to achieve this forms the first part of the DSSSL standard, which defines a General Language Transformation Process (GLTP) that can take objects in an SGML-encoded data tree and associate them with objects that can be used by a formatter. This transformation uses an advanced, SGML-knowledgeable, query language to identify the relationships between objects making up the source document and those making up the output of the transformation process. The relationship between SGML elements, their attributes, the file storage entities that contain them and the entities and elements they contain can all be identified and mapped, as appropriate, for output. This means that attributes in the source document can create new elements or entities in the output document, and that elements or storage entities can be used to control attribute (property) setting in the structure passed to the text formatter.

The DSSSL GLTP performs a role that TeX does not address. It will allow you to take an SGML-coded document and turn it into a format that is suitable for processing by a known set of TeX macros. By allowing, for example, structurally related cross-references (e.g., see Chapter 4) to be resolved prior to formatting, with the appropriate formatting properties being generated in response to the type of reference, DSSSL should be able to reduce the amount of work that needs to be done during formatting significantly.

The DSSSL document formatting Semantic Specific Process (SSP) will:

- provide a set of formatting properties that have internationally acceptable formal definitions of their meaning,
- provide a model for describing areas into which data is to be positioned or flowed, and
- provide a method for describing which area, or type of area, each object in the structured document should be placed into.

The first stage of the document formatting SSP consists of describing the relationships between the various areas that make up a page, and the properties of each area, in an area definition. The second stage consists of describing how the elements that make up the GLTP output tree are to be 'flowed' into the areas described in the area definition. Figure 1 shows the relationship between these processes.

Rather than invent a completely new language to define the relationships between objects in the various transformations, DSSSL has been developed as an extension to IEEE's Scheme variant of LISP, produced by MIT. The adoption of this advanced AI language offers a number of important advantages. In particular LISP is an object-oriented language that is ideally suited to querying data structures and trees. Any location in a document tree can be described in Scheme as a list of the objects that make up the tree, e.g.:

```
(document body (chapter 4)
(section 5) (subsection 2) (para 3)).
```

Scheme provides a compact, but clearly defined, set of functions that can be used to manipulate and transform object lists. DSSSL provides the additional functions needed to describe the relationships between Scheme processes and SGML constructs, e.g.: `(query-tree *root* '(EL "p")`.

Another key consideration in the choice of Scheme was the simplicity of its powerful recursive processing features. Formatting is largely a recursive process, and Scheme can simplify the expression of recursive processes.

While the exact form of the Scheme constructs is still under discussion the following examples will give you some idea of the form the final language will probably take. A typical area definition for the running head of a left-hand page might be:

```
(area-def left-header
   '(rep glyph)
   (set-properties
    (area-type line)
    (origin (point 7pi 0.5in))
    (x-extent 39pi)
    (y-extent 1pi)
```

```
+------+ +-------------+ +--------+ +-------------+ +------+
|Source| | Association | | Output | |    Flow     | | SGML |
| DTD  | |Specification| |  DTD   | |Specification| |Output|
+------+ +------+------+ +--------+ +-------------+ +------+
       |        |         |      |         |         |
       v        v         v      v         v         ^
       |        |         |      |         |         |
      +---------------------+   +---------------------+
      |  General Language   |   |  Semantic Specific  |
      |   Transformation    |   |       Process       |
      |   Process (GLTP)    |   |        (SSP)        |
      +---------------------+   +---------------------+
       |        |         |      |         |         |
       ^        ^         v      ^         ^         v
       |        |         |      |         |         |
+--------+ +-----------+ +--------+ +-----------+ +---------+
| Source | | Resource  | | Output | |   Area    | |Formatted|
|Instance| |Definitions| |Instance| |Definitions| |  Image  |
+--------+ +-----------+ +--------+ +-----------+ +---------+
```

Figure 1: The Modules of DSSSL.

```
    (placement-path-start
    (point 0 baseline-offset))
    (placement-path-end
    (point 39pi baseline-offset))
    (align centre)
    (word-spaces (18 15 18))
    ; there are 54 units to 1 em
    (letterspace (6 6 6))
    (hyphenation-allowed #f)
    (font-name "/Monotype/Helvetica/Medium")
    (point-size 10pt)
    (set-width 10pt)
    ((case usc)
    ; Uppercase for capitals
       - small caps for lowercase
   ))
```

while a typical flow specification could have the form:

```
((title title-p)
    ((wide-text single-column-text-area)
      (set-properties
       (prespace 1in)
       (postspace 3pi)
       (font-name heading-font)
       (font-size 36pt)
       (path-separation 42pt)
       (align centre)
       (last-line centre)
       (hyphenation-allowed #f)
       (cond (count
            (children (line title title-p))
             < 1)
             (generated-text
              "Unnamed Report")
      )) )
      (left-header left-page)
```

In this example the title element on the title page flows into both an area for holding wide text lines in a single colum text area and into the running header for left-hand pages. For the title page the formatting properties are applied as part of the flow definition because, in this case, the area being used to contain the text is a general-purpose one whose default typesetting properties are not suitable for the type of text about to be poured into it. DSSSL's ability to qualify the way in which areas are formatted dependent on the contents that are flowed into it provide a type of functionality provided by few formatters. Note particularly the inclusion of a standard Scheme condition expression that acts as a trap for cases where no data has been entered into the title field of the report. In this case the left-hand running head will be blank, but the title page will at least have a heading (Unnamed Report).

One of the interesting points of discussion among DSSSL developers is the relationship between the various sets of properties that can be used to control processing. The way in which text should be presented to users can be defined as:

- using attributes defined in the source instance;
- using attributes defined as part of the output GLTP;
- through SSP-defined properties assigned to areas in the area definition; or
- through SSP-defined properties attached to the flow specification.

Which of these should take precedence, and in what order should they be applied? Personally I feel that attributes in the source, or those created as part of the transformation process, should be able to override properties defined as part of an area definition, but should they also override any properties specified while defining how output objects should be flowed into areas? There seems to be no clear-cut answer to this dilemma, and it looks as if the DSSSL team will need to define a precedence order as part of the standard based on gut feeling, unless someone can come up with a convincing case for a particular approach. (Any offers?)

How Can DSSSL Be Used in Conjunction with TEX?

The DSSSL GLTP process can be used to transform SGML-encoded files into forms suitable for processing by existing TEX macro sets, such as those provided by LATEX. In such cases there is little need to associate area definitions with the GLTP transformation as this is the function of the TEX macros. However, in the longer term, it would be advantageous if we could map the way in which DSSSL describes areas, and the properties used to describe the required output, directly into TEX. Hopefully this will not prove too difficult a task, especially given the transformational power provided by Scheme.

The work currently being done on LATEX3 should make it easier to use TEX as the output process of the DSSSL process. As I understand it, one of the aims of LATEX3 is to allow attributes/properties to be used to control the way in which macros process the associated text. If we can find a way to map the properties defined in DSSSL to equivalent functions in TEX it should be possible to provide simple transformation algorithms that will turn the output of the DSSSL GLTP process into appropriate LATEX3 macro calls. While we are still a number of years from being able to do this, now is the time to plan how this should be achieved, before DSSSL or LATEX3 are completed. For this reason the DSSSL team is keeping a close eye on what is happening in the TEX world.

As DSSSL is designed to be used with a wide range of formatters, including those based on TEX, it is important that the way in which DSSSL properties can be interpreted in a TEX environment be carefully studied prior to publication of the final standard. For this reason it is important that the TEX community track the changes that will be published in the second draft of the international standard when it is published later this year.

TeX and SGML: A Recipe for Disaster?

Peter Flynn
University College
Cork, Ireland
Internet: `pflynn@curia.ucc.ie`

Abstract

The relationship between TEX and SGML (Standard Generalised Markup Language, ISO 8879) has often been uneasy, with adherents to one system or the other displaying symptoms remininscent of the religious wars popular between devotees of TEX and of word-processors.

SGML and TEX can in fact coexist successfully, provided features of one system are not expected of the other. This paper presents a pilot program to test one method of achieving such a cohabitation.

Introduction

For many years, SGML and its relationship with TEX has been a frequent topic of presentation and discussion. Network users who read the TEXhax digest and the Usenet newsgroup `comp.text.tex` will be familiar with the sometimes extensive cross-postings to the `sgml-l` mailing list and the `comp.text.sgml` newsgroup. Two extremes are apparent in the misunderstandings: that SGML is some kind of desktop publishing (DTP) system; and that TEX or are exclusively for structured documentation. Such problems highlight the lack of information about the design of either system, as available to the novice, but also reveal the capabilities and limitations of both systems.

In fact, there is a parlous level of understanding about both TEX and SGML even in the printing and publishing industry, where one would expect a more sophisticated degree of understanding: in this author's personal hearing, so-called experts from major publishing houses have criticised TEX's 'lack of fonts' and SGML's 'lack of font control'.

It is perhaps worth emphasising the difference at this stage, for the non-expert, in that TEX is a typographic system principally for the creation of beautiful books (Knuth, 1984) (but also other printed documents: it is intended for putting marks on paper) and SGML (Goldfarb, 1990) is the international standard for describing the structure of documents (intended for document storage and control, which could, of course, include typesetting as one of many possibilities).

Publishing: the view from outside

A recent article (Beard, 1993) quotes John Watson, London Editorial Director of Springer-Verlag:

> We can use LATEX files, which many of our authors of books or papers with complex maths find convenient, but if they need serious editing, it's so expensive we have to mark up hardcopy and send it back to the author to make the changes. *TEX and LATEX are only a stop-gap. SGML hasn't really reached our authors yet.* What's really needed is a WYSIWYG system that's as universal as TEX, preferably in the public domain so all our authors and freelances can use it, and easy for subject specialists to edit on screen. And of course the output should be Linotron- as well as Postscript-compatible. **(Emphasis added.)**

This view of the world expresses an attitude common in the publishing field, that editing TEX is difficult, that the nature of TEX is impermanent, and that the only goal of all writing is for it to be printed on paper. While SGML has indeed 'not reached our authors yet', that is hardly the fault of SGML, when editing systems for handling SGML are readily available for most platforms.

The speaker's desires are very laudable, however much one may agree or disagree with the implied benefits of WYSIWYG systems, in that the software should be universal, easy to use and in the public domain. The speaker's complaints, however, deserve further analysis.

Editing. The speaker seems here to be confusing two aspects of the technical editorial process: mathematics editing and copy editing (editing text for production), both of which have to date been perceived as matters for the specialist, as those who use TEX in a professional pre-production capacity with publishers as clients have long recognised.

In the confusion, sight has been lost of the fact that editing a file of TEX source code need be no more of a problem than editing any other kind of file, if an adequate macro structure is provided, and it is probably less of a problem the better structured the text is. If the publisher's authors are unable or unwilling to adhere to the very straightforward guidelines put out by most publishers, it would appear a little ingenuous to blame TEX for their deficiencies.

There are large numbers of literate and numerate graduates with sometimes extensive TEX experience: if (as seems to be implied) editing may now be entrusted to authors, a publisher has little excuse for not employing some of these graduates on non-specialist editorial work. It is, however, as unnerving to hear publishers so anxious to encourage authors to undertake pre-press editing as it would be to hear them encourage non-mathematicians to undertake mathematical editing: it is precisely because the authors do not normally possess the specialist knowledge to do this that the work is handled by in-house or contract editors. The mechanics of editing a TEX document are not especially difficult, given proficiently-written macros, and there are some crafty editor programs around to assist this task. Training courses in elementary TEX abound, so if a publisher is serious about cutting pre-press costs by using TEX, the way lies open.

The typographic skill resides in implementing the layout: taking the typographer's specifications and turning them into TEX macros to do the job, ideally leaving the author and subsequent editor with as little trouble as possible to get in the way of the creative spirit. The implementation of design is, however, increasingly being left to the author, who may understandably resent having to undertake what is usually seen as a task for the publisher, and who may be ill-equipped to perform this task (Fyffe, 1969), especially if a purely visual DTP system is being used.

Impermanence. TEX has been around for nearly 15 years, longer than any other DTP system, and quite long enough for the mantle of impermanence to be shrugged off: there is no other system which can claim anywhere near that level of stability and robustness. However, the present writer would be among the first to disclaim any pretensions on the part of TEX to being the final solution to a publisher's problems (although properly implemented it has no difficulty in seeing off the competition). It is difficult, however, to understand what TEX is supposed to be a stop-gap for, because the logical conclusion a reader might draw from the quotation above is that SGML is some kind of printing system, which it is not, although it can be used for that purpose (for example, in conjunction with something like TEX).

Printing as a goal. WYSIWYG TEX systems exist for both PCs and Macintosh platforms, if a user feels compelled to see type springing into existence prematurely. There are also similar editors for SGML, ranging from the simple to the sophisticated. The misconception seems to be that printing on paper is always going to be the goal of the writer and the publisher, but even if we accept this goal as the current requirement, there appears to be no reason why both TEX and SGML cannot be used together to achieve this.

The increasing importance being attached to hypertext systems, especially in academic publishing, is amply evidenced by the presentations at scholarly conferences, for example (Flynn, 1993) the recent meeting of the Association for Literary and Linguistic Computing and the Association for Computing and the Humanities. While paper publication will perhaps always be with us, alternative methods are of increasing importance, and systems such as SGML are acknowledged as providing a suitable vehicle for the transfer and storage of documents (Sperberg-McQueen and Burnard, 1990) requiring multiple presentations.

Software development. Before we leave this analysis, it is worth asking if publishers who are seeking an easy-to-use, widely-available, public-domain WYSIWYG-structured editor would be prepared to back their demands with funding for the development of such a system. Organisations such as the Free Software Foundation are well-placed to support and coordinate such an effort, and there are ample human resources (and considerable motivation) in the research and academic environment to achieve the target.

Document Type Disasters

The newcomer to SGML is often perplexed by the apparent complexity of even simple Document Type Definitions (DTDs, which specify how a document is structured). Although there are several excellent SGML editors on the market, many users are still editing SGML in a plain file editor with perhaps the use of macro key assignments to speed the use of tags and entity references. Worse, the task of getting the document printed in a typographic form for checking by proofreaders who are unfamiliar with SGML can present a daunting task without adequate software.

While we have said that such software is readily available, there are two inhibiting factors: cost and complexity. Although we are now beginning to see wordprocessor manufacturers take an interest in SGML (WordPerfect, for example), the impecunious researcher or student is still at a disadvantage, as WYSIWYG software for SGML is still expensive for an individual.

The problem of complexity is not easily solved: designing a document at the visual level of typography is already understood to be a specialist task in most cases, and designing a document structure, which is a purely conceptual task, without visual representation, is at a different level of abstraction. However, document structure design is not normally the province of the publisher's author, and should not affect the author's use of a structured-document editor, once the initial concept has been accepted.

Into print. The `comp.text.sgml` newsgroup repeatedly carries requests from intending users for details of available editing and printing software, which are usually answered rapidly with extensive details. The low level of SGML's public image (the 'quiet revolution' (Rubinsky, 1992)) indicates one possible reason why the system is still regarded with misgivings by some people.

There have been several attempts in the past to develop systems which would take an SGML instance and convert its text to a TEX or LATEX file for printing. The earliest appears to have been `Daphne`, developed in the mid 1980s by the *Deutsche Forschungsnetz* in Berlin, and the most recent is `gf` (`comp.text.sgml`, 4.6.1993) from Gary Houston in New Zealand (available from the Darmstadt `ftp` server). Several other programs exist, including some written in TEX itself, but the principal stumbling-block seems to be the desire to make the program read and parse the DTD so that the instance can be interpreted and converted accordingly.

A DTD contains information principally about the structure of the documents which conform to it, rather than about its visual appearance. (It is of course perfectly possible to encode details on visual appearance in SGML, but this is more the province of the analyst or historian, who wishes to preserve for posterity the exact visual nature of a document.) The DTD is used to ensure conformance, often by an editor while the document is being written or modified, or by a parser (a program which checks the syntax and conformity of an instance to its DTD). Given the easy availability of various versions of a formal SGML parser (`sgmls`, from various `ftp` archives), there seems to be little point in embedding that process again in a formatter. Indeed, one conversion system reported to this author takes the route of using `sgmls` output as its input.

Through all these systems, however, runs the thread that somewhere in the SGML being used must reside all the typographical material needed to make the conversion to TEX (or indeed any typographical system) a one-shot process. As has been pointed out, this implies that the author or writer using SGML to create the document must embed all the necessary typographical data in the instance. Yet this is entirely the opposite of the natural use of SGML, which is to describe document *structure* or *content*, not its appearance. Predicating typographic matters ties the instance to one particular form of appearance, which may be wholly irrelevant.

Style and content. One of TEX's strongest features is that of the style file, a collection of macros to implement a particular layout or format. In particular, where this uses some form of standardised naming for the macros, as with LATEX or `eplain`, the portability of the document is greatly enhanced. A single word changed in the `documentstyle` and the entire document can be re-typeset in an entirely different layout, with (usually) no further intervention.

The convergence of SGML and TEX for the purposes of typesetting brings two main advantages: the use of TEX's highly sophisticated typesetting engine and the formally parsed structure of the SGML instance. In such a union, those elements of the DTD which do have a visual implication would migrate to a macro file, in which specific coding for the visual appearance *of the current edition* could be inserted, and the SGML instance would migrate to a TEX or LATEX file which would use these macros.

In this way, we would avoid entirely the predication of form within the SGML: it becomes irrelevant for the author to have to be concerned with the typographic minutiæ of how the publication will look in print (although obviously a temporary palliative can be provided in the form of a WYSIWYG editor). We also avoid tying the instance to any one particular layout, thus enabling the republication (or other reuse) in a different form at a later date with a minimum of effort.

The most undemanding form of conversion is thus one where the appearance is completely unreferenced in the SGML encoding. This means that the publisher (or typesetter) has all the hooks on which to hang a typographic implementation, but is not restricted or compelled to use any particular one of them.

A pilot program: `sgml2tex`

The author's own pilot attempt at this form of conversion can be seen in the SGML2TeX program, available by anonymous `ftp` from `curia.ucc.ie` in `pub/tex/sgml2tex.zip`. This was developed in PCL (a language written explicitly for high speed development on the *80n86* chips): WEB should probably be the basis for a future version.

The program reads an SGML instance character by character, and converts all SGML tags into TEX-like control sequences, by removing the < and > delimiters and prepending '`\start`' or '`\finish`' to the tag name. Attributes are similarly treated, within the domain of the enclosing element, and with their value given in curly braces as a TEX macro argument. Entity references are converted to simple TEX control sequences of the same name.

The output from the program is a `.tex` file and a `.sty` file. The `.tex` file contains an '`\input`' of the `.sty` file at the start, and also a '`\bye`' at the end; otherwise it is merely a representation of the instance in a form digestible by TEX or LATEX. The `.sty` file contains a null definition of every element, attribute and entity encountered in the instance. Thus the fragment

```
    prepend '<tt>&bsol;start</tt>'
```

becomes

```
    prepend
'\startTT{}\bsol{}start\finishTT{}'
```

in the `.tex` file, with the following definitions in the `.sty` file:

```
\def\startTT{}
\def\finishTT{}
\def\bsol{}
```

All line-ends, multiple spaces and tabs in the instance are condensed to single space characters.

It must be made clear that this pilot is not a parser: it does not read any DTD and has no understanding of the SGML being processed, although a planned rudimentary configuration file will allow a small amount of control over the elimination of specific elements where no conversion is desired. There is also no capability yet for handling any degree of minimisation, so all markup must be complete and orthogonal (as many parsers and editors already have the capability to output such non-minimised SGML code, this should not cause any problems). As the DTD is not involved, the instance being converted must therefore also have passed the parsing stage: it is the user's responsibility to ensure that only validly-parsed instances are processed. Additionally, no attempt has been made to support scientific, mathematical or musical tagging, as this is outside the scope of the pilot.

As it stands, therefore, the output file is a valid TEX file, although trying to process it with null definitions in the `.sty` file would result in its being treated as a single gigantic paragraph. However, editing the `.sty` file enables arbitarily complex formatting to be implemented: the present document (`http://curia.ucc.ie/tlh/curia/doc/achallc.html`) is a simple example.

Conclusions

The pilot program certainly *is* a stop-gap, being severely limited: there are many other related areas where SGML design, editing, display and printing tools are still needed. There is still no portable and widespread public-domain dedicated SGML editor such as would encourage usage (although an SGML-sensitive modification for `emacs` exists and the interest of WordPerfect has been noted). Although SGML import is becoming available for some high-end DTP systems, migration and conversion tools are still at a formative stage.

One particular gap is highlighted by the need for a program to assist the user in building a DTD, with a graphical interface which would show the structure diagrammatically, so that permitted and prohibited constructs can be analysed, and a valid DTD generated.

SGML has now passed the phase of 'new product' and is on its way to greater acceptance, but the real disaster would be for it to become an isolated system, unrelated to other efforts in computing technology. This will only be avoided by the concerted efforts of users and intending users in demanding software which can bridge the gaps.

Bibliography

Beard J. "The art and craft of good science", *Personal Computer World*, page 350, June 1993.

Fyffe C. *Basic Copyfitting*, London: Studio Vista, page 60, 1969.

Goldfarb C. The SGML Handbook, OUP, 1990.

Knuth D.E. *The TEXbook*, Addison-Wesley, 1984.

Rubinsky Y. "The Quiet Revolution", keynote speech, SGML92 Conference, October 1992.

Sperberg-McQueen C.M. and Burnard L., (eds). *Guidelines for the Encoding and Interchange of Machine-Readable Texts*, Draft version 1.1, ACH/ACL/ALLC, Chicago & Oxford, 1990.

An Abstract Model for Tables

Xinxin Wang
Department of Computer Science, University of Waterloo
Waterloo, Ontario N2L 3G1, Canada
internet: wang@watdragon.uwaterloo.ca.

Derick Wood
Department of Computer Science, University of Western Ontario
London, Ontario N6A 5B7, Canada
Internet: dwood@csd.uwo.ca.

Abstract

We present a tabular model that abstracts a wide range of tables. We abstract the logical structure of tables, rather than their presentational form. The model can be used to guide the design and implementation of tabular editors and formatters. In addition, the model is formatter independent; it can be used to direct the formatting of tables in many typesetting systems, including TEX.

Introduction

Although tables are widely used in daily life to convey information in a compact and convenient form, tabular processing is one of the most difficult parts of document processing, because tables are more complex than other textual objects. The separation of the logical and layout structures of documents is widely used in many document formatting systems (Lamport (1985); Quint and Vatton (1986); and Reid (1980)). It enables authors to focus on the manipulation of the logical structure of a document. The layout structure is determined by the formatting systems based on style specifications; thus, high quality typeset documents can be produced with little or no help from typographers. Tabular formatting is, however, the weak link in most formatting systems. The main reason is that the tabular models used in many systems (Beach (1995); Biggerstaff et al. (1984); Cameron (1989); Lamport (1985); and Lesk (1979)) are **presentation dependent;** that is, the models describe tables based on their presentational form. In other words, it is the user's responsibility to design the geometric arrangement of tabular components. Some systems (Improv Handbook (1991) and Vanoirbeek and Coray, eds. (1992)) use presentation-independent models for tables that are based on their logical structure; however, the models fall short in that they are made with specific environments in mind. The strength of our model is that it is not tied to any specific realization and it can be viewed as an abstract data type. One other drawback of most tabular systems is that the tabular operations that are provided are too weak to manipulate tables based on the logical relationships among tabular components.

We are currently developing a tabular composition system based on this model, which can be used as a front end for LATEX tables.

In this paper, we first summarize the main characteristics of tables, and then present our model. To conclude the presentation, we compare our model with Vanoirbeek's model and also discuss the influence of our model on the design and implementation of a tabular composition system.

The Characteristics of Tables

The Oxford English Dictionary defines a table as: "an arrangement of numbers, words or items of any kind, in a definite and compact form, so as to exhibit some set of facts or relations in a distinct and comprehensive way, for convenience of study, reference, or calculation". This definition summarizes the characteristics of a table using three different aspects: content, form and function.

The content of a table. The content of a table is a collection of interrelated items, which can be numbers, text, symbols, figures, mathematical equations, or even other tables. In most tables, these items can be divided into two classes based on their function in the table: **entries,** which are facts of any kind that we present in a table, and **labels,** which we use to locate the entries. The logical relationships among the items of a table are the associations among labels and entries. Each entry is associated with a set

Table 1: The average marks of CS351(1991-1992).

Mark / Year Term	Assignments		Exams		Final marks
	Ass1	Ass2	Midterm	Final	
1991					
Winter	85	80	70	73	74
Summer	78	79	65	70	70
Fall	82	80	–	80	80
1992					
Winter	83	78	72	75	75
Summer	80	76	–	78	78
Fall	76	74	60	80	72

Table 2: The average marks of CS351(1991-1992).

Term / Year / Mark	Winter		Summer		Fall	
	1991	1992	1991	1992	1991	1992
Assignments						
Ass1	85	83	78	80	82	76
Ass2	80	78	79	76	80	74
Exams						
Midterm	70	72	65	–	–	60
Final	73	75	70	78	80	80
Final Marks	74	75	70	78	80	72

of labels; for example, in Table 1, entry 85 is associated with labels 1991, Winter, Assignments and Ass1. The items and the logical relationships among them provide the **logical structure** of a table, which is the primary information conveyed by the table and which is independent of its presentational form.

We can describe the logical structure of a wide range of tables in this way: first, we group the labels into n **categories** such that in each category labels are organized in a tree structure, and then we associate each entry with one, or more, n-element sets of label sequences where each label sequence is the catenation of labels on the path from the root to a leaf in a category. For example, the labels of Table 1 can be grouped into three categories:

Year = {1991, 1992},
Term = {Winter, Summer, Fall}, and
Mark = {Assignments, Exams, Final marks}.

In category Mark, there are two subcategories:

Assignments = {Ass1, Ass2} and
Exams = {Midterm, Final}.

Entry 85 is associated with a 3-element set of label sequences: {Year.1991, Term.Winter, Mark.Assignments.Ass1}; Entry 76, which appears in the table twice, is associated with two 3-element sets of label sequences: {Year.1992, Term.Fall, Mark.Assignments.Ass1} and {Year.1992, Term.Summer, Mark.Assignments.Ass2}.

The form of a table. The content of a table must be presented in some form and on some medium. Usually, tables are presented as a row-column structure on a two-dimensional plane, such as paper or screen. The presentational form of a table consists of two components: the topological arrangement and the typographic specification. The topological arrangement is an arrangement of the table components in two-dimensional space such that the logical structure of the table is clearly conveyed; for example, where to put the labels and entries and how to order the labels in a category. The typographic specification is a group of formatting attributes for rendering tabular data and the graphic objects that are used to outline the topological arrangement, such as the font type for entries, the line style for rules, and so on. The content of a table can be presented with different topological arrangements and different typographic specifications. For example, Tables 1 and 2 are two different presentations for a three-category table. Although the row-column structure is a familiar and natural form for tabular presentation, tables may also be presented in other forms, such as the bar graph, the pie graph, and so on.

The function of a table. The main function of a table is to convey data and its relationship in a compact and convenient way.

The Tabular Model

In our opinion, a tabular composition system should allow users to be mainly concerned about the logical structure of tables; they should leave the presentational form to a high-quality tabular formatting system that requires little or no user intervention. A tabular model for such a system should possess the following characteristics:

- it can be used to abstract a wide range of tables;
- it is presentation independent; that is, it captures the logical structure of tables and ignores any topological and typographic attributes; and
- it includes a group of operations that support tabular manipulation.

```
(D1,  { (d11, { (d112, ∅),
                (d113, ∅),
              }),
        (d12, ∅),
        (d13, ∅),
      })
```

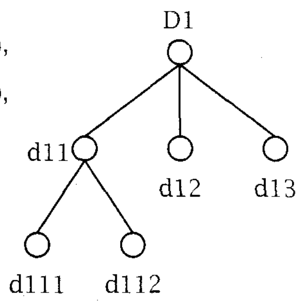


Figure 1: The relationship between a labeled domain and its tree.

We specify our tabular model with mathematical notions so as to avoid the representational structure and the implementation details. Therefore, the model can also be viewed as an abstract data type; that is, an abstract table and a set of operations.

Terminology. We first define some terminology before we give the definition of an abstract table.

A **labeled set** is a label together with a set. We denote a labeled set as $(label, set)$.

A **labeled domain** is defined inductively as follows:

1. A labeled empty set $(L, \emptyset)$ is a labeled domain.
2. A labeled set of labeled domains is a labeled domain.
3. Only labeled sets that are obtained with rules 1 and 2 are labeled domains.

A labeled domain can be represented by an unordered tree of labels. Figure 1 presents the relationship between a labeled domain and its tree. Each node in the tree represents a labeled domain. For convenience, we will use the tree of a labeled domain to explain some concepts that are related to labeled domains. If a labeled domain $D = (L, S)$, we use $L[D]$ to denote the label L, of D, and $S[D]$ to denote the set S of D.

A **label sequence** is either one label or the catenation of multiple labels separated with the symbol '.'. With the tree of Figure 1, $D1$ and $D1.d11.d111$ are two examples of label sequences. Operation $\odot$ takes a $label$ and a set of label sequences $\{s_1, \cdots, s_n\}$ as operands and its result is a set of label sequences such that

$$label \odot \{s_1, \cdots, s_n\} = \{label.s_1, \cdots, label.s_n\}.$$

The **frontier** of a labeled domain D is the set of external nodes of the tree of D. It is denoted by $F[D]$ and is defined inductively as

1. $\{label\}$, if $D = (label, \emptyset)$;
2. $label \odot (\bigcup_{x \in S} F[x])$, if $D = (label, S)$.

With the labeled domain of Figure 1, $F[D1] = \{D1.d11.d111, D1.d11.d112, D1.d12, D1.d13\}$.

A **frontier item** of label domain D is any member of F[D]. It is the label sequence on a path from the root to an external node in the tree of labeled domain D.

An **item** of a labeled domain D is any prefix of a frontier item of D; it is the label sequence on a path from the root to a node in the tree of D. With the labeled domain of Figure 1, $D1$, $D1.d11$, $D1.d11.d111$, $D1.d11.d112$, $D1.d12$ and $D1.d13$ are all items of the labeled domain $D1$. An item is actually the label sequence on the path from the root to a node in the tree of the labeled domain D. We use an item to identify its associated node.

If i is an item, we use $LD[i]$ to denote the labeled domain represented by the associated node of i and $P[i]$ to denote the item that identifies the direct parent of the associated node of i. For example, with the labeled domain of Figure 1, if $i = D1.d11.d112$, then $LD[i]$ is the labeled domain $(d112, \emptyset)$ and $P[i] = D1.d11$. If $LD[i] = (L, S)$, we also use $L[i]$ to denote the label L, $S[i]$ to denote the set S, and $F[i]$ to denote the frontier of $LD[i]$.

The **dimension** of a labeled domain $D = (L, S)$ is denoted by $Dim[D]$; it is the number of elements in S. With the labeled domain of Figure 1, $D1 = (D1, \{d11, d12, d13\})$; thus, $Dim[D1] = 3$. We say that two items i and j are in the same dimension of D if and only if both the associated nodes of i and j are in a child subtree of the tree of D. For example, with the labeled domain of Figure 1, $D1.d11$ and $D1.d11.d111$ are in the same dimension of $D1$, but $D1.d11$ and $D1.d12$ are not.

For $n > 1$, an **n-set** is a set of n elements. For two sets A and B, $A \otimes B$ is the set of all 2-sets that consists of one element of A and one element of B. $A \otimes B$ is similar to, yet different from, $A \times B$, the Cartesian product of A and B. It is similar in that we take all pairs of elements, one from A and one from B; it is different because we obtain unordered pairs, rather than ordered pairs. It is **unordered Cartesian product**.

We now apply $\otimes$ to labeled domains as follows. It takes a labeled domain $D = (L, S)$ as operand and it results in a set in which each element contains $Dim[D]$ frontier items, each of which identifies an external node of a labeled domain in S; that is,

$$\begin{aligned} \otimes D &= \emptyset, \quad \text{if } S = \emptyset \\ &= \{L \odot \{t_1, \cdots, t_n\} \mid t_i \in F[D_i], 1 \le i \le n\}, \\ &\quad \text{if } S = \{D_1, \cdots, D_n\}. \end{aligned}$$

Table 3: A three-category table.

D1 \ D2 \ D3		d31		d32
		d311	d312	
d11	d21	e1	e2	e5
	d22	e3		
	d23	e4		
d12	d21	e6	e7	e9
	d22			
	d23		e8	

For example, with the labeled domain of Figure 1,

$$\begin{aligned}\otimes D1 \;=\; & \{\{D1.d11.d111,\ D1.d12,\ D1.d13\},\\ & \{D1.d11.d112,\ D1.d12,\ D1.d13\}\}.\end{aligned}$$

The definition of an abstract table. An **abstract table** consists of three elements: a labeled domain, a set of entries, and a function from a set of n-element sets of frontier items (n is the dimension of the labeled domain) to the set of entries. It can be formally defined by a tuple (D, E, δ), where

- D is a labeled domain
- E is a set of entries
- δ is a partial function from $\otimes D$ onto E

We use a labeled domain D to describe the category structure of a table, the dimension of a labeled domain corresponds to the number of categories of the table, and each labeled domain in $S[D]$ corresponds to a category. We use a function to describe the logical associations among labels and entries. Using this model, Table 3 can be abstracted by (D, E, δ), where

$$\begin{aligned}D = (D, \{&(D1,\ \{(d11, \emptyset),\ (d12, \emptyset)\}\ ),\\ &(D2,\ \{(d21, \emptyset),\ (d22, \emptyset),\ (d23, \emptyset)\}),\\ &(D3,\ \{(d31,\ \{(d311, \emptyset),\ (d312, \emptyset)\}),\\ &\qquad (d32, \emptyset)\ \}\\ &\ )\\ &\}\\ ),\quad &\end{aligned}$$

$E = \{e1, e2, e3, e4, e5, e6, e7, e8, e9\}$, and

$$\begin{aligned}
\delta(\{D.D1.d11,\ D.D2.d21,\ D.D3.d31.d311\}) &= e1;\\
\delta(\{D.D1.d11,\ D.D2.d21,\ D.D3.d31.d312\}) &= e2;\\
\delta(\{D.D1.d11,\ D.D2.d22,\ D.D3.d31.d311\}) &= e3;\\
\delta(\{D.D1.d11,\ D.D2.d22,\ D.D3.d31.d312\}) &= e3;\\
\delta(\{D.D1.d11,\ D.D2.d23,\ D.D3.d31.d311\}) &= e4;\\
\delta(\{D.D1.d11,\ D.D2.d21,\ D.D3.d32\}) &= e5;\\
\delta(\{D.D1.d11,\ D.D2.d22,\ D.D3.d32\}) &= e5;
\end{aligned}$$

Table 4: A two-category table.

D1 \ D2	S1				S2
	S11		S12	S13	
	S111	S112			
L1	e1	e2	e3	e4	e5
L2	e6	e7	e8	e9	e10
L3	e11	e12	e13	e14	e15

$$\begin{aligned}
\delta(\{D.D1.d11,\ D.D2.d23,\ D.D3.d32\}) &= e5;\\
\delta(\{D.D1.d12,\ D.D2.d21,\ D.D3.d31.d311\}) &= e6;\\
\delta(\{D.D1.d12,\ D.D2.d21,\ D.D3.d31.d312\}) &= e7;\\
\delta(\{D.D1.d12,\ D.D2.d22,\ D.D3.d31.d312\}) &= e7;\\
\delta(\{D.D1.d12,\ D.D2.d23,\ D.D3.d31.d312\}) &= e8;\\
\delta(\{D.D1.d12,\ D.D2.d21,\ D.D3.d32\}) &= e9;\\
\delta(\{D.D1.d12,\ D.D2.d22,\ D.D3.d32\}) &= e9;\\
\delta(\{D.D1.d12,\ D.D2.d23,\ D.D3.d32\}) &= e9.
\end{aligned}$$

Basic operations for abstract tables. We define a basic set of operations for tabular editing. These operations are divided into four groups: operations for categories, items, labels, and entries. For each operation, we first give its name and the types of its operands and result, and then explain its semantics informally.

Category operations. There are two operations for categories.

The operation **Add_Category** adds a new category to a table. It takes a table $T = (D, E, \delta)$ and a labeled domain D_m as operands, and returns a new table $T' = (D', E', \delta')$ such that:

(1) D' is produced by inserting D_m into the set of D;
(2) the entry set E' is the same as E;
(3) δ' maps any $fs \in \otimes D'$, which contains an element $L[D].f$ such that f is a frontier item of D_m, to $\delta(fs - \{L[D].f\})$. For example, if T is Table 4, Add_Category($T, D3$), where $D3 = (D3, \{(T1, \emptyset), (T2, \emptyset)\})$, produces Table 5.

The operation **Remove_Category** removes a category from a table. It takes a table $T = (D, E, \delta)$ and an item d_i (which must identify an element of the set of labeled domain D) as operands, and returns a new table $T' = (D', E', \delta')$ such that:

(1) D' is produced by deleting the labeled domain $LD[d_i]$ from the set of D;

Table 5: After adding a new category to Table 4.

D2		S1				S2
		S11		S12	S13	
D3	D1	S111	S112			
T1	L1	e1	e2	e3	e4	e5
	L2	e6	e7	e8	e9	e10
	L3	e11	e12	e13	e14	e15
T2	L1	e1	e2	e3	e4	e5
	L2	e6	e7	e8	e9	e10
	L3	e11	e12	e13	e14	e15

Table 6: After removing a category from Table 5.

D2	S1				S2
	S11		S12	S13	
D3	S111	S112			
T1	e1 e6 e11	e2 e7 e12	e3 e8 e13	e4 e9 e14	e5 e10 e15
T2	e1 e6 e11	e2 e7 e12	e3 e8 e13	e4 e9 e14	e5 e10 e15

Table 7: After inserting an item to Table 4.

D2	S1					S2
	S11		S12	S13	S14	
D1	S111	S112				
L1	e1	e2	e3	e4		e5
L2	e6	e7	e8	e9		e10
L3	e11	e12	e13	e14		e15

Table 8: After deleting an item from Table 4.

D2	S1			S2
	S11		S13	
D1	S111	S112		
L1	e1	e2	e4	e5
L2	e6	e7	e9	e10
L3	e11	e12	e14	e15

(2) E' is a set in which each element is $\{\delta(fs \cup \{L[D].k_1\}), \cdots, \delta(fs \cup \{L[D].k_m\})\}$ where fs is any element of $\otimes D'$ and $k_1, \cdots, k_m$ are all frontier items of $LD[d_i]$;

(3) δ' maps any $fs \in \otimes D'$ to set $\{\delta(fs \cup \{L[D].k_1\}), \cdots, \delta(fs \cup \{L[D].k_m\})\}$. For example, if T is Table 5, then Remove_Category$(T, D1)$ produces Table 6.

Item operations. There are four operations for items.

The operation **Insert_Item** inserts a labeled tree to a category. It takes a table $T = (D, E, \delta)$, one of its items p (which cannot be D) and a labeled domain C as operands and returns a new table $T' = (D', E', \delta')$ such that:

(1) D' is produced by inserting C into the tree of D such that C will be a child of $LD[p]$;

(2) E' is the same as E;

(3) if p is a frontier item of D, then δ' will map every element $fs \in \otimes D'$ which contains $p.f$ such that f is a frontier item of C, to $\delta((fs - \{p.f\}) \cup \{p\})$; otherwise, δ' on these elements is undefined; for other $fs \in \otimes D'$, $\delta'(fs)$ is the same as $\delta(fs)$. For example, if T is Table 4, Insert_Item$(T, D.D2.S1, C)$, where $C = (S14, \emptyset)$, produces Table 7.

The operation **Delete_Item** deletes a labeled tree from a category. It takes a table $T = (D, E, \delta)$ and one of its items i (which cannot be D or any item that identifies a child of D) as operands, and returns a new table $T' = (D', E', \delta')$ such that:

(1) D' is produced by removing the labeled domain $LD[i]$ from D;

(2) E' is produced by removing all entries that are not mapped from any element in $\otimes D'$ by δ;

(3) if the old parent of i, i.e, $P[i]$, becomes a frontier item, then δ' on any $fs \in \otimes D'$, which contains $P[i]$, is undefined; for other $fs \in \otimes D'$, $\delta'(fs)$ is the same as $\delta(fs)$. For example, if T is Table 4, Delete_Item$(T, D.D2.S1.S12)$ produces Table 8.

The operation **Move_Item** moves a subtree to a new place within a category. It takes a table $T = (D, E, \delta)$ and two of its items c and p that

Table 9: After moving an item in Table 4.

D2 / D1	S1		S11		S2
	S12	S13	S111	S112	
L1	e3	e4	e1	e2	e5
L2	e8	e9	e6	e7	e10
L3	e13	e14	e11	e12	e15

are in the same dimension of D (p cannot be a descendant of c) as operands, and returns a new table $T' = (D', E', \delta')$ such that:

(1) D' is produced by moving labeled domain $LD[c]$ to be a child of labeled domain $LD[p]$;

(2) E' is the same as E;

(3) δ' maps any $fs \in \otimes D'$ which contains item $p.t$ where t is a frontier item of $LD[c]$ to $\delta((fs - \{p.t\}) \cup \{P[c].t\})$, and if the old parent of c , i.e, $P[c]$, become a frontier item of D', then δ on any $fs \in \otimes D'$, which contains $P[c]$, is undefined; for other $fs \in \otimes D'$, $\delta'(fs)$ is the same as $\delta(fs)$. For example, if T is Table 4, Move_Item$(T, D.D2.S1.S11, D.D2)$ produces Table 9.

The operation **Copy_Item** duplicates a subtree in a category. It takes a table $T = (D, E, \delta)$, two of its items c and p that are in the same dimension of D, and a label l as operands, and returns a new table $T' = (D', E', \delta')$ such that:

(1) D' is produced by copying labeled domain $LD[c]$ to be a child of labeled domain $LD[p]$ and assigning label l to the new labeled domain copied from $LD[c]$;

(2) E' is the same as E;

(3) if c is a frontier item of D, then δ' maps any $fs \in \otimes D'$ which contains item $p.l$ to $\delta((fs - \{p.l\}) \cup \{c\})$, otherwise, δ' maps any $fs \in \otimes D'$ which contains item $p.l.t$ such that $c.t$ is a frontier item of D to $\delta((fs - \{p.l.t\}) \cup \{c.t\})$; for other $fs \in \otimes D'$, $\delta'(fs)$ is the same as $\delta(fs)$. For example, if T is Table 4, Copy_Item$(T, D.D2.S1.S112, D.D2.S1, S14)$ produces Table 10.

Label operations. There are two operations for labels.

The operation **Put_Label** assigns a new label to a labeled domain. It takes a table $T = (D, E, \delta)$, one of its items i, and a label l as operands, and returns a new table by assigning the label l to labeled domain $LD[i]$.

Table 10: After copying an item in Table 4.

D2 / D1	S1					S2
	S11		S12	S13	S14	
	S111	S112				
L1	e1	e2	e3	e4	e2	e5
L2	e6	e7	e8	e9	e7	e10
L3	e11	e12	e13	e14	e12	e15

The operation **Get_Label** takes a table $T = (D, E, \delta)$ and one of its items i and returns the label of i.

Entry operations. There are two operations for entries.

The operation **Put_Entry** associates a new entry with a set of frontier items. It takes a table $T = (D, E, \delta)$, an entry e and a number of frontier items $f_1, \cdots, f_{Dim[D]}$ such that $\{f_1, \cdots, f_{Dim[D]}\}$ must be an element of $\otimes D$. It returns a new table by putting entry e into table T such that the new function maps $\{f_1, \cdots, f_{Dim[D]}\}$ to e. If the old entry mapped from $\{f_1, \cdots, f_{Dim[D]}\}$ is not mapped from any other element in $\otimes D$, it will be deleted from E.

The operation **Get_Entry** returns the entry that is associated to a set of frontier items. It takes a table T and a number of frontier items $f_1, \cdots, f_{Dim[D]}$ such that $\{f_1, \cdots, f_{Dim[D]}\}$ must be an element of $\otimes D$ as operands, and returns the entry that is mapped from $\{f_1, \cdots, f_{Dim[D]}\}$.

Conclusions

We have presented a tabular model that, although it is not a universal model, can be used to abstract a wide range of tables. This model is presentation independent because it abstracts only the logical structure of multi-dimensional tables and excludes any topological and typographic attributes. This characteristic makes it possible to design a tabular composition system in such a way that users are mainly concerned about the logical structure of tables, and the layout structure of a table is determined by the system based on style specifications. In this way, we can manipulate and format tables in a uniform way like other textual objects.

Our model is similar to Vanoirbeek's model (Vanoirbeek and Coray, eds., 1992), although we derived it independently. The major difference between these two models is the way to specify the logical structure of a table. In Vanoirbeek's model, the logical structure of a table is modeled by a tree with additional links: a table contains a set of logical dimensions and a set of items (entries); the logical dimensions include rubrics (labels) which may themselves contain subrubrics; links are used to represent the connections between items and rubrics. The main reason for this representation mechanism is to ensure that the table representation conforms with the hierarchical structured document representation used in the host system Grif (Quint and Vatton, 1986). In our model, the logical structure of a table is specified mathematically; it avoids the representational structure and implementation details. Our model is not tied to any specific environment; thus, we can develop a tabular composition system based on this model that can be used to direct the formatting of tables in different typesetting systems. Another difference is that the operations for rearranging the category structure and maintaining the logical relationships among labels and entries in Vanoirbeek's model and its Grif implementation are weaker than those in our model; for example, we can move and copy all labels in a subtree of a category and their associated entries.

Acknowledgements

We thank Darrell Raymond for his support and careful reading of a preliminary version of this paper.

Bibliography

Improv Handbook. Lotus Development Corporation, Cambridge, MA, 1991.

Beach, R. J. *Setting Tables and Illustrations with Style*. PhD thesis, University of Waterloo, Waterloo, Ontario, Canada, May 1985. Also issued as Technical Report CSL-85-3, Xerox Palo Alto Research Center, Palo Alto, CA.

Biggerstaff, Ted J., D. Mack Endres, and Ira R. Forman. "TABLE: Object Oriented Editing of Complex Structures". In *Proceeding of the 7th International Conference on Software Engineering*, pages 334-345, 1984.

Cameron, J. P. A Cognitive Model for Tabular Editing. Technical Report OSU-CISRC-6/89-TR 26, The Ohio State University, Columbus, OH, June 1989.

Lamport, Leslie. *LaTeX: A Document Preparation System*. Addison-Wesley, Reading, MA, 1985.

Lesk, M. E. "`tbl`—A Program to Format Tables". In *UNIX Programmer's Manual*, volume 2A. Bell Telephone Laboratories, Murray Hill, NJ, 7th edition, January 1979.

Quint, Vincent and Irène Vatton. "Grif: An Interactive System for Structured Document Manipulation". In *Text Processing and Document Manipulation, Proceedings of the International Conference*, pages 200-312, Cambridge, UK, 1986. Cambridge University Press.

Reid, Brian K. *Scribe: A Document Specification Language and its Compiler*. PhD thesis, Carnegie-Mellon University, Pittsburgh, PA, October 1980. Also issued as Technical Report CMU-CS-81-100, Carnegie-Mellon University.

Vanoirbeek, Christine. "Formatting Structured Tables". In C. Vanoirbeek & G. Coray, editor, *EP92(Proceedings of Electronic Publishing, 1992)*, pages 291-309, Cambridge, UK, 1992. Cambridge University Press.

Developing a Multi-Windowing Environment for Research Based on TeX

Michel Lavaud
C.N.R.S.
GREMI, Université d'Orléans,
45067 ORLÉANS Cedex (France)
Internet: `lavaud@centre.univ-orleans.fr`

Abstract

We have devised an experimental program, A^S^TeX, which provides an easy to use multi-window environment adapted to research work. It runs on any PC and is built on the top of a commercial all-in-one software (Framework). It endows it with scientific capabilities by coupling it to emTeX, Maple, Fortran and other scientific software. It allows for the easy modification of the structure of large multi-author documents and the performance of numerical and formal computations from the document. It adds a hypertext file manager, a preprocessor of LaTeX structure, hypertext help and hypertext archiving of messages, among others. It can use the multitasking capabilities of Desqview or OS/2.

Many of the functions of A^S^TeX could be implemented also on the top of other existing software (commercial or public domain), provided they are endowed with an internal programming language which is powerful enough. We hope this could be done with GNU *emacs*.

Several commercial WYSIWYG scientific word-processors on PCs are now able to produce very nice output. Since the advantage of TeX is diminishing as concerns quality of output, some leading TeXperts have concluded that it is becoming too old, and have proposed creating a New Typesetting System from scratch, which would incorporate all aspects that are missing from TeX.

On the other hand, more and more scientists have access to international networks, and they are now using TeX as a language in the *linguistic* sense of the term, i.e., as a *means of communication.* This implies that TeX must remain stable in time as much as possible, for it to be able to fulfill this communication function.

We suggest that keeping TeX unchanged, as desired by many users, is not incompatible with building easy-to-use and powerful TeX-based software, as desired by TeXperts. This can be done by improving front ends and back ends to TeX and making them cooperate together via a multitasking OS.

In this article we describe a program, A^S^TeX, that we have written and that illustrates this point of view. It might provide — we hope — some guidelines for future developments in this direction.

Existing interfaces to TeX

In his article about the Future of TeX (Taylor, 1992), Philip Taylor described how painful it was to use TeX in the early eighties. Although he assured that TeX users enjoy this way of working... for those who do not, there are now several user-friendly public domain interfaces to TeX and related software, that make its use much easier! The first one is the AUCTeX Lisp package for GNU *emacs.* It is extremely powerful since it is based on the complete version of *emacs*; this requires 386-based PCs and big hard-disks, and preferably OS/2 or Unix. Another interface that runs under OS/2 is TeXpert, by Johannes Martin.

For DOS users, there is the very nice interface TeXshell, by J. Schlegelmilch. Its latest version (2.5.2) is particularly useful for all users, since there is now on-line help on LaTeX in English. It is also public domain, and very user-friendly. A new one, 4TeX (W. Dol, et al., 1993), appeared very recently. It uses the shareware programs 4DOS and QEDIT. It seems very nice too. Our interface A^S^TeX uses the commercial program Framework, and optionally Desqview or OS/2.

Finally, there are also several commercial scientific word-processors that are able to edit mathematical equations in WYSIWYG mode and are able to export them in TeX form (see Lavaud, 1991 and 1992 for some references).

This shows that, even on PCs, the Lion must not be afraid any more of the Mouse (Siebenmann, 1992)...

Motivations for writing A^S^TEX

In the early history of computer science, programs were written on sheets of paper by researchers, typed on punched cards by specialized typists, and submitted to the computer by an operator. When teletype terminals became available, all scientists began to type and run their computer programs themselves because this allowed them to gain much time, despite the fact that they did the work of three people.

Many researchers still write their scientific articles by hand, and have them typed by secretaries. This can take a very long time, especially for articles with many complicated formulae. It seems reasonable to expect that, if software adapted to research work becomes available, all scientists will also type their articles themselves, because this will allow them to gain much time, as with teletype terminals. In an earlier article (Lavaud, 1992), we argued that, for a software to be adapted to scientific work:

- It must allow the user to *display* and *modify* easily the structure of the document, to ensure that even very long multi-author documents will be coherent and logically organized;
- it must allow for the performance of everyday research tasks *from the document* (numerical and formal computations, management of the files created or received in the research process, etc.);
- it must be TEX-based.

The first point implies that a document cannot be just a sequence of characters typed inside one or a few windows; it must be a *tree* whose leaves are windows that contain coherent blocks of information of various nature (paragraph of text, worksheet of numerical results, database, computer program, numerical result, e-mail, illustration, etc.).

Overview of A^S^TEX's possibilities

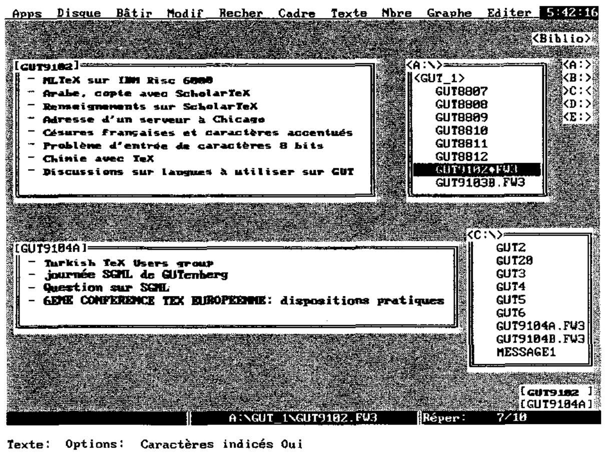
We enumerate here the main possibilities of A^S^TEX. Some are developed in more detail below. Others are detailed in Lavaud, 1991 and 1992 and in references therein.

- **Hypertext file manager**:
  - Immediate access to thousands of files through hierarchy of explicit titles.
  - Easy modification of the structure of very big multi-author documents.
- **Scientific computations**:
  - Numerical (e.g., Fortran): compilation / execution run directly from text of the document.
  - Formal (e.g., Maple): results automatically included in text, worksheets, databases.
  - Live links to data files.
  - Interdependent worksheets.
- **Scientific text processing**:
  - Mathematical and chemical formulas displayed with a single keystroke.
  - Preprocessor of LATEX structure.
  - Cut and paste from hypertext help into the document.
  - Automatic generation of environments.
  - Creation of LATEX tables from worksheets or databases of formulas.
- **Tool Box**:
  - External DOS (and UNIX) tasks can be run from a customizable Tool Box.
- **Electronic mail**:
  - Hypertext archiving of messages.
  - Automatic extraction of messages from files issued from discussion lists.
  - Local archiving of information about ftp and archie servers to speed up connections.
- **Hypertext help** for A^S^TEX, LATEX, emTEX, Ghostscript, graphs used in physics ...

The hypertext file manager of A^S^TEX

Writing a multi-author scientific book from scratch, (or collating results of a research team regularly over several years) is not only the matter of typing text with a scientific word-processor. When you create with colleagues a document that will at the end have several hundred pages, and that will make reference to hundreds of files (articles, chapters of book, numerical results, computer programs, input and output data, electronic illustrations, electronic mail...), it is very important to be able to navigate easily, in a *logical* way, in the document and in the files that are related to it, during the whole process of its creation. You need a file manager which allows you to classify and archive your files in a *structured* way, so that you can retrieve them easily, with a few keystrokes, regardless of who created them.

Usual file managers. With usual file managers (those of Framework 3, of Windows 3.1, etc.), you access a file from its *physical location* on disks: you have first to remember on which disk it is, then in which directory. Then, you have to scroll among a set of files, most of which are not pertinent to your document. Moreover, as names of files and directories are limited to eight characters (with MS/DOS), they are not very explicit in general, and it may be very difficult to retrieve a file that was created or received a long time ago (see Figure 1). Even worse, files created by colleagues and pertaining to the document may have been moved without their alerting you.


Apps Disque Bâtir Modif Recher Cadre Texte Nbre Graphe Editer 5:42:16
<Biblio>
<A:> <B:> >C:< <D:> <E:>

[GUT9102]
- MLTeX sur IBM Risc 6000
- Arabe, copte avec ScholarTeX
- Renseignements sur ScholarTeX
- Adresse d'un serveur à Chicago
- Césures françaises et caractères accentués
- Problème d'entrée de caractères 8 bits
- Chimie avec TeX
- Discussions sur langues à utiliser sur GUT

<A:\>
<GUT_1>
GUT8807
GUT8808
GUT8809
GUT8810
GUT8811
GUT8812
GUT9102.FW3
GUT9103B.FW3

[GUT9104A]
- Turkish TeX Users group
- journée SGML de GUTenberg
- Question sur SGML
- 6EME CONFERENCE TEX EUROPEENNE: dispositions pratiques

<C:\>
GUT2
GUT20
GUT3
GUT4
GUT5
GUT6
GUT9104A.FW3
GUT9104B.FW3
MESSAGE1

[GUT9102]
[GUT9104A]
A:\GUT_1\GUT9102.FW3 Réper: 7/10

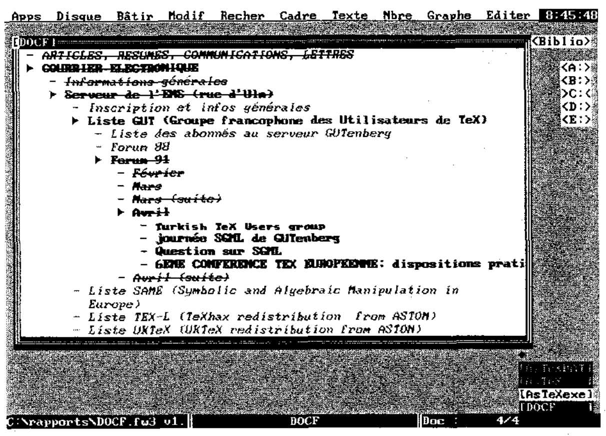
Texte: Options: Caractères indicés Oui

Figure 1: File retrieval with the file manager of Framework 3, illustrated with e-mail received from the GUTenberg discussion list (French TeX Users Group): names of archive files are not explicit, files are scattered over several disks.

The file manager of A^{S}T$_{E}$X. For A^{S}T$_{E}$X, a document is a *set of files related logically* and accessible from one of them (the master file) by loading into linked windows. The set of files has a *tree structure*, and each file is itself a tree whose leaves are *objects* of various nature (linked windows, texts, databases, worksheets, graphics, computer programs...). The files may be on different media and be created/modified by several people on a network.

With the hypertext file manager of A^{S}T$_{E}$X, a file is accessed from its *logical location* in the document, not from its physical location on disk. Each file related to the document is retrieved from a hierarchy of *explicit titles.* This way of accessing files has many advantages, among which (see Lavaud, 1991 for more details):

- The way to retrieve a file from the document *remains unchanged* when the file is moved physically to another place for some reason (the directory is too crowded, the local hard disk is full,...).
- Only the files *pertinent to the document* are displayed and accessible from it. The files that are unrelated are not displayed.
- *Modifying the structure* of a document is very easy and very fast, because files are reorganized logically in the document, not physically on disk(s).
- A file can be accessed from several documents, in different ways, i.e., with different hierarchies of titles.
- Data (e.g., computer programs, numerical results, etc.) can be accessed as *live links* (i.e., the latest version of the data file is automatically loaded) or stored as *backups* in parent files.
- One has *several levels of backup* for linked files.
- You are automatically informed of new files added to the document by colleagues, without them having to tell you.


Apps Disque Bâtir Modif Recher Cadre Texte Nbre Graphe Editer 8:45:48
<Biblio>
<A:> <B:> >C:< <D:> <E:>
[AsTeXexe]

[DOCF]
- ~~ARTICLES, RESUMES, COMMUNICATIONS, LETTRES~~
▸ ~~COURRIER ELECTRONIQUE~~
  - ~~Informations générales~~
  ▸ ~~Serveur de l'ENS (rue d'Ulm)~~
    - Inscription et infos générales
    ▸ Liste GUT (Groupe francophone des Utilisateurs de TeX)
      - Liste des abonnés au serveur GUTenberg
      - Forum 88
      ▸ ~~Forum 91~~
        - ~~Février~~
        - ~~Mars~~
        - ~~Mars (suite)~~
        ▸ ~~Avril~~
          - Turkish TeX Users group
          - journée SGML de GUTenberg
          - Question sur SGML
          - 6EME CONFERENCE TEX EUROPEENNE: dispositions prati
        - ~~Avril (suite)~~
    - Liste SAME (Symbolic and Algebraic Manipulation in Europe)
    - Liste TEX-L (TeXhax redistribution from ASTON)
    - Liste UKTeX (UKTeX redistribution from ASTON)

[DOCF]
C:\rapports\DOCF.fw3 v1. DOCF Doc : 4/4

Figure 2: File retrieval with the hypertext file manager of A^{S}T$_{E}$X, illustrated with the same example as in Figure 1. e-mail is retrieved from a hierarchy of titles.

Word processing with A^{S}T$_{E}$X

The general philosophy of A^{S}T$_{E}$X is to display interactively only *global formatting* of text, and to use LaTeX commands for *local formatting.* These are considered as encapsulated in small blocks of information that are stored into linked files. A^{S}T$_{E}$X deals with the organization of these blocks through organization of linked windows on screen, and it allows the author to forget completely about local formatting commands. He is just reminded of the contents of the blocks through their titles, and he can concentrate on the important part of his work, that is on the *logical connection* of the various components of his document and on the *scientific computations* that are related to it.

WYSIWYG or not WYSIWYG? That is the question! When speaking of WYSIWYG scientific editors, one thinks automatically of interactive editing of mathematical formulas. In an earlier article (Lavaud, 1992), we explained that there are many interactive equation editors able to *export* mathematical equations in TEX, but that, to be really useful, they ought to be able also to *import* such equations and, more generally, any TEX or LATEX file. And this is much more difficult, since this means that the equation editor must contain almost all the capabilities of the TEX compiler.

With ASTEX, mathematical equations and most local formatting commands are supposed to be written in native TEX. Some simple local commands that Framework is able to display, such as italics, bold, indices and exponents, can be translated automatically by ASTEX. It provides also on-line hypertext help and a multi-level assistance in typing LATEX code, in particular by generating automatically environments from a hierarchical menu (see Lavaud, 1992).

Previewing portions of text. In the absence of a satisfactory WYSIWYG editor able to import TEX and LATEX files, a good front end to TEX must be able at least to preview any portion of text with one or a few keystrokes (Siebenmann, 1992). This has been possible for quite a long time with *emacs*. This is possible also with ASTEX (see Figure 3). It is further possible to preview the text contained in a selected subset of windows, as appears in Figure 4.

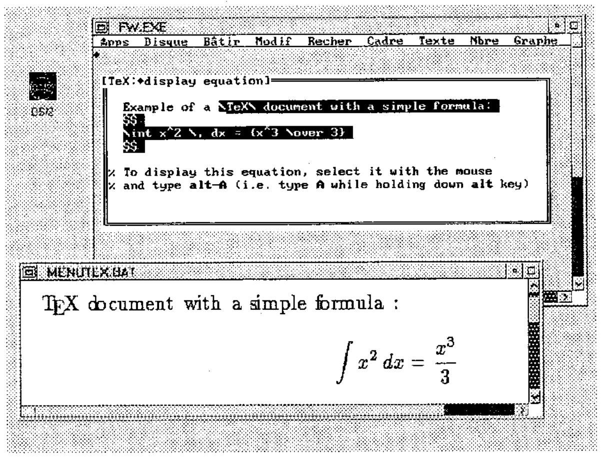
As Laurent Siebenmann has emphasized, the mechanism is very simple, but it seems very underused. For example, the question "*How can I transform my Wordperfect files to TEX or LATEX*" is asked very often on the net. Now, with Wordperfect, mathematical equations are typed in the eqn language and are debugged exactly as indicated in Figure 3. So, instead of trying to transcode from eqn to TEX, it would be much more efficient to write and debug equations directly in TEX. A program, written in the programming language of Wordperfect, that would implement the above mechanism would certainly solve many problems. More generally, this mechanism could be implemented very easily into many word-processors, so that files in native TEX could be typed and debugged from these word-processors, instead of being translated by an external program, so that users accustomed to a given word-processor can take advantage of TEX from within their favorite editor. So, although the mechanism is fairly trivial, let us describe it in some detail for the PC.

The editor needs only to be able to save a selected portion of text into a file, shell to DOS and run an external program. A prolog and a trailer have to be added to the selected text: this can be done either inside the editor, if it is able to concatenate chains, or during the shell by adding `\input prolog` and `\input trailer` at the beginning and at the end of the file containing the text, with the DOS copy instruction.

This results in previewing a portion of text by switching from source text in full screen to the previewer in full screen, and back again to the editor. A more elaborate way is to display code and result simultaneously as in Figure 3. This is obtained by coupling the preceding mechanism to the multitasking properties of Desqview or OS/2. With Desqview for example, instead of running the previewer directly during DOS shell, you have to run the shareware utility *dvexec*, telling it to create a child window and to run the previewer in it. This is done by running a batch program containing a line such as:

```
dvexec c:\dv\tp-pif.dvp
```

where `tp-pif.dvp` is a file created by Desqview, which contains the parameters to run the TEX previewer.



```
[TeX:♦display equation]

  Example of a \TeX\ document with a simple formula:
  $$
  \int x^2 \, dx = {x^3 \over 3}
  $$

% To display this equation, select it with the mouse
% and type alt-A (i.e. type A while holding down alt key)
```

TEX document with a simple formula :

$$\int x^2\,dx = \frac{x^3}{3}$$

Figure 3: Displaying a portion of text with ASTEX: the text selected with the mouse or the key arrows, is displayed automatically in a child window by typing a single keystroke (`alt-A`).

The prolog and trailer attached to the current document have to be stored somewhere, in external files or inside separate windows, according to the capabilities of the editor used. The prolog must contain all the necessary definitions. For example, if we want to preview the chemical formula:

```
$$
\ethene{$CH_2 OH$}{$R^2$}{$R^3$}{$R^4$}
$$
```

with the ChemTeX package, the prolog must contain at least the instructions:

```
\documentstyle[chemtex]{article}
\begin{document}
```

The prolog must also contain personal macros that are used in the document. The trailer must contain at least the `\end{document}` instruction for LaTeX, or `\bye` for TeX.

With ASTEX, since many individual documents may be stored inside the master document, each may have a special prolog/trailer. These are stored in windows that immediately precede/follow the subtree that contains the text of the document.

Preprocessor of LaTeX structure. When you write a long document, you have to modify its structure very often. Local commands, such as `\it`, `\indent`, mathematical formulas, etc. remain unchanged. But global commands such as `\chapter` or `\section` must usually be modified: for example, if a long section is becoming too big and has to be transformed into a chapter, all `\subsection` commands must be transformed into `\section`, etc.... This may require modifying many LaTeX sectioning commands, which may be very error-prone if these are scattered among several files.

With ASTEX, you do not have to modify sectioning commands because you never write any of them. Any modification in the structure of the document is made within Framework, and sectioning commands are automatically generated by ASTEX from the tree of the Framework document.

Implementing some functionalities of SGML parsers. SGML is the ISO standard for document description. It is designed specifically to enable text interchange (van Herwijnen, 1990). Although SGML is not very well adapted to everyday research work, many of its ideas are very important and of general scope, and can be implemented fruitfully into TeX-based software. For example, an important function of SGML parsers is to ensure that a document has no chapter inside a section. This possibility is not forbidden by LaTeX: if we want to write "Hello everybody!" in large letters in the middle of a paragraph, it is possible to do it by including the instruction:

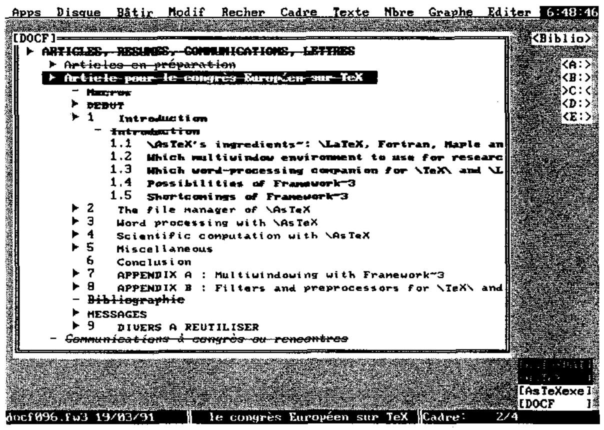
```
\section*{Hello everybody!}
```

With ASTEX, it is impossible to create an ill-structured document, for two reasons:

1. Sectioning commands are generated automatically from the tree of the document (cf. preceding section);
2. Writing sectioning commands in text is inhibited: if we type `\section` in the text, it is automatically erased, and the message "*Instruction \section forbidden*" is displayed on the screen.

Therefore ASTEX plays the role, at the front end level, that is fulfilled by SGML parsers at the back end level.

Exporting a LaTeX document. Let us consider the document of Figure 4 (which corresponds to the article by Lavaud, 1991):


Apps Disque Bâtir Modif Recher Cadre Texte Nbre Graphe Editer
6:48:46
<Biblio>
<A:>
<B:>
>C:<
<D:>
<E:>
[AsTeXexe]
[DOCF]

```
[DOCF]
  ▶ ~~ARTICLES, RESUMES, COMMUNICATIONS, LETTRES~~
    ▶ ~~Articles en préparation~~
    ▶ ~~Article pour le congrès Européen sur TeX~~
      - ~~Macros~~
      ▶ DEBUT
      ▶ 1   Introduction
          - ~~Introduction~~
          1.1  \AsTeX's ingredients~: \LaTeX, Fortran, Maple an
          1.2  Which multiwindow environment to use for researc
          1.3  Which word-processing companion for \TeX\ and \L
          1.4  Possibilities of Framework~3
          1.5  Shortcomings of Framework~3
      ▶ 2   The file manager of \AsTeX
      ▶ 3   Word processing with \AsTeX
      ▶ 4   Scientific computation with \AsTeX
      ▶ 5   Miscellaneous
        6   Conclusion
      ▶ 7   APPENDIX A : Multiwindowing with Framework~3
      ▶ 8   APPENDIX B : Filters and preprocessors for \TeX\ and
      - ~~Bibliographie~~
      ▶ MESSAGES
      ▶ 9   DIVERS A REUTILISER
    - ~~Communications à congrès ou rencontres~~
docf096.fw3 19/03/91   le congrès Européen sur TeX   Cadre:   2/4
```

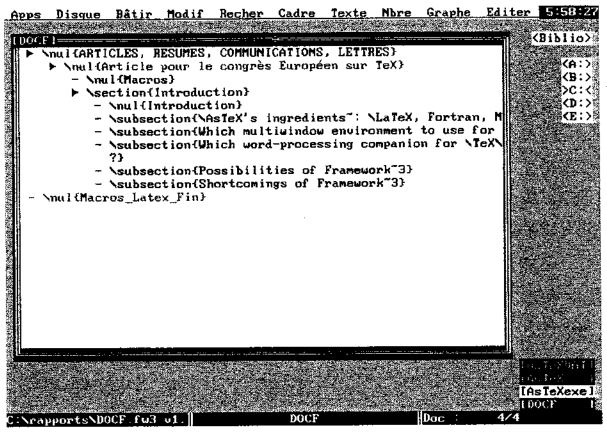
Figure 4: Example of an article, in the Table of Contents mode.

When ASTEX is asked to create a LaTeX file from it, it generates the document of Figure 5. We see that ASTEX does not blindly export the whole document: a lot of windows have been eliminated. When the document is displayed as in Figure 4 (title of first section in boldface, other titles in normal characters), it is configured for debugging only the first section of the article.

To illustrate further the great flexibility of ASTEX in creating LaTeX documents, let us just indicate that the master document of Figure 4 manages several thousand files; nevertheless when going from the state of Figure 4 to Figure 5, ASTEX has exported only a small part of one file linked to the document; and it could have exported as easily small selected (non-consecutive) parts of several files.

The Tool Box of ASTEX

ASTEX contains a Tool Box that automates access to general internal or external resources, independent of the current document. *Internal resources* are, for example, agenda, alarm, frequently used databases...). *External resources* are in general batch programs, to run external PC programs in a customized environment or to send UNIX requests to the server. All resources are accessible from the



```
▶ \nul{ARTICLES, RESUMES, COMMUNICATIONS, LETTRES}
   ▶ \nul{Article pour le congrès Européen sur TeX}
      - \nul{Macros}
      ▶ \section{Introduction}
         - \nul{Introduction}
         - \subsection{\AsTeX's ingredients~: \LaTeX, Fortran, M
         - \subsection{Which multiwindow environment to use for
         - \subsection{Which word-processing companion for \TeX\
           ?}
         - \subsection{Possibilities of Framework~3}
         - \subsection{Shortcomings of Framework~3}
- \nul{Macros_Latex_Fin}
```

Figure 5: Document with LATEX sectioning commands, generated by ASTEX from the document of Figure 4.

Tool Box exactly as any item in a hierarchical system of menu. They are activated by pointing to an explicit title instead of typing the name of a program.

Porting ASTEX to other software

ASTEX has been developed on the top of Framework because, at the time when the project began (1990), this commercial program was the most suited to our purpose, while public domain editors available on PCs were not powerful enough. In particular, *emacs* was not available in its complete version.

Many functions of ASTEX can be implemented in other software, either PD or commercial, provided it has a powerful enough programming language. We enumerate the main possibilities offered by Framework that are used by ASTEX, to indicate the prerequisites for such a porting.

Why use Framework?

- Framework offers a *hierarchical* multiwindowing system for the three basic applications:
  - editor of text,
  - spreadsheet,
  - database manager.
- It has a very powerful programming language, which allows us to program very complex applications. This language is identical in all applications (when using specialized programs, you have to learn several different languages).
- It is ideally complementary to TEX: each application is much more rudimentary than a specialized program (e.g., the spreadsheet module as compared to Excel), but most possibilities that are missing are added by its coupling with TEX, and many more are added that may not exist in the specialized program. For example, mathematical formulas cannot be written in cells of Excel, while this can be done with Framework + TEX (of course, this could also be done by coupling Excel to TEX).
- Telecommunications can be done in a window, with the possibility to cut and paste text to and from other windows containing text, worksheet or database.
- The three basic applications run much faster than an equivalent set of programs under Windows 3.1, and the multi-windowing system for the basic applications, combined with multi-windowing facilities of Desqview or OS/2, is much more powerful than that which can be obtained with Windows 3.1.

Framework also has several other advantages: it runs on any PC, occupies only about 2 Mbytes on disk, and it has some interesting built-in possibilities (spell-checker, synonyms, mailing, etc.).

Porting ASTEX to GNU *emacs*. Our dearest wish would now be to port ASTEX to GNU *emacs*. Indeed, the complete version of *emacs*, with its Lisp-like programming language, has been ported to OS/2. This is still a limitation, because this requires a PC386 and large disks, but hardware prices are going down very fast and older models will disappear soon.

Porting ASTEX to *emacs* would be desirable for many reasons. First, it is public domain, well-supported and widely used. Second, since Framework is a commercial program, some of its shortcomings cannot be solved. For example, although Framework is mouse-based, no control of the mouse is provided by its programming language. Since the code of Framework is not public domain, this makes programming the use of the mouse with ASTEX very difficult. Many other problems cannot be solved neatly for the same reason. For example, we could only forbid typing `\section {}` but not `\section{}`, because only the space character is considered as an end of word, in the automatic substitution function of Framework. This fairly stupid limitation could be solved in a few lines of code with a public domain editor.

Conclusion

We have proved, by building the program ASTEX, that it is possible not only to make the use of TEX and related software easy on low-cost PCs, but also to build a powerful multi-window environment that is adapted to scientific research and based on TEX.

For A^S^TeX to be useful in practice (not only as a model), it ought now to be ported to other more widely used commercial software and above all to public domain editors, in particular to *emacs*.

Bibliography

Dol, Wietse, Eric Frambach, and Maarten van der Vlek, *MAPS 93.1*, pages 53 - 57, 1993.

van Herwijnen, Eric, *Practical SGML*, Kluwer, 1990.

Lavaud, Michel, *EuroTeX'91 Conference Proceedings*, pages 93 - 116, 1991.

Lavaud, Michel, *EuroTeX'92 Conference Proceedings*, pages 307 - 330, 1992. Reprinted in *MAPS 93.1*.

Siebenmann, Laurent, *EuroTeX'92 Conference Proceedings*, pages 43 - 52, 1992.

Taylor, Philip, *EuroTeX'92 Conference Proceedings*, pages 235 - 254, 1992. Reprinted in *MAPS 93.1*.

A Versatile TEX Device Driver

Minato Kawaguti
Fukui University, 9-1, Bunkyo-3, Fukui, 910 Japan
Internet: `kawaguti@ilmpsl.fuis.fukui-u.ac.jp`

Abstract

A new TEX DVI driver was developed for Canon's LaserShot family (supported by LIPS III) and its twin, the LBP family (supported by CaPSL III/IV) laser-beam printers. High grade "publisher quality" TEX output using a conventional simple (that is, non-PostScript) laser-beam printer has been achieved: [1] four categories of fonts, (i) pk, (ii) printer-resident scalables, (iii) scalable fonts for emulating typesetters, and (iv) PostScript Type 1 format, can be used; [2] missing pk fonts may be generated automatically from either mf files or PostScript fonts; [3] various character ornamentations, such as: (i) gray scale inking, (ii) filling the glyphs with patterns, (iii) character outlining, (iv) drop-shadowing, (v) character shading, and (vi) black/white reversing, have been incorporated; and [4] insertion of figures into TEX documents through encapsulated PostScript files is possible despite the fact that non-PostScript printers are used.

Why non-PostScript Printer Driver?

A new TEX DVI driver was developed for Canon's LaserShot family (supported by LIPS III and IV control language) and its twin, the LBP family (supported by CaPSL III/IV control language) laser-beam printers. The main objective was to achieve high grade "publisher quality" TEX output with a conventional simple (that is, non-PostScript) laser-beam printer.

To be sure, the existing DVI drivers for PostScript printers, *e.g.*, Tom Rokicki's *dvips*, give excellent output. Nevertheless there exists, particularly among academic and research communities where people rely heavily on TEX, the persistent demand for a DVI driver with superior performance applicable to non-PostScript printers as well.

From the standpoint of TEX users, the majority of the pages of documents to be printed daily are in 'pure' TEX format. PostScript figures within a document, if any, occupy only a small fraction of the pages. In these circumstances, 'native' control language of the printers is equally or better suited, offering certain advantages in economy, and possibly in the printing speed and prolonged service life of the printer to the extent that complexity of the control software is simpler.

TEX is widely accepted as a prevalent norm for the generic means of exchanging documents across the computer networks. In the due course of matured sophistication of the TEX usage, however, the fine quality typesetting demanded by some documents is by now somewhat contradicted by the underlying implicit premise that the recipient is assured of reproducing the received documents exactly as the original copy the sender dispatched. This confidence, one of the characteristic virtues of TEX from the earliest days, can be reaffirmed much further by letting any available printer print practically any TEX document, irrespective of whether fancy typefaces are used, or some figures are embedded within the pages. All in all, it would be beneficial to the entire TEX community if we could enhance printing capabilities of all types of laser-beam printers.

Current Printer Environment

The current average TEX printer environment might be surmised through a recent episode of an international conference. The organizing committee of *Computing in High Energy Physics '92*, which was held at Annecy, France, in September 1992, announced to all the authors of the contributing papers that the camera-ready copy for the proceedings be submitted by typesetting with the network-distributed LATEX style file designed for the proceedings, which supposedly unified the font of the entire proceedings to Times-Roman and its derivatives.

Of the 197 papers actually published in the proceedings, a quick visual inspection reveals that those which conform to the Times-Roman or its near look-alikes were 130, while the remaining 67 were evidently alien to the specified font. The fact that only 66% could attain the style unification depicts the still haunting problem of availability of fonts other than the Computer Modern.

With regard to non-alphabetic languages of which Japanese is one, the issue of fonts is more serious. The significant cost overhead associated with the Japanese PostScript fonts, and also the penalty of speed of PostScript printers in the past in dealing with these fonts, could have influenced the current proliferate use, among TEX users, of non-PostScript printers equipped with resident scalable fonts for the language.

Extended Functionality

The major objective of the development of the DVI driver to be described (henceforth denoted as the *Driver* for short), therefore, is to bestow the non-PostScript printer with the following extended features so that it can cope with the variety of unorthodox needs likely to be encountered in printing TEX documents written by some of the demanding users.

Figures. Insertion of figures in a TEX text should be performed in the standard manner. The figure, described in a separate file in the PostScript form, should be allocated at the specified location in the prescribed size. The image figure based on a bitmap file should also be included in the TEX document.

Fonts. It should be possible to mix the fonts in a multitude of formats, including pk fonts and standard PostScript fonts.

Ornamentation. For broadening the area of TEX application, it is highly desirable to add ornamental power of decorative expression beyond the standard TEX capabilities. Among others the following features are particularly noteworthy:

1. character highlighting for instruction manuals;
2. inclusion of half-tone shaded boxes for generating various forms;
3. artistic fonts for cover pages of books and manuals;
4. distinct taste or appeal of formal letters through the choice of adequate fonts; and
5. eye-catching special effects design for posters.

Basic Driver Action

The way the *Driver* prints the information in a DVI file can be summarized as follows:

1. The *Driver* translates the information in a DVI file into the 'native' control codes of the printer.
2. To insert a figure, the content of the specified PostScript file is interpreted by a slightly modified version of GNU's *Ghostscript* interpreter as a filter.
3. It deals with a large variety of fonts by obtaining the glyph data from the files on a multitude of media, including compact discs.
4. Various kinds of character ornamentations are performed for special effects while printing by TEX. The *Driver* receives the command specific to the printer through the 'special' command of TEX.
5. It manages the buffer memory of the printer while downloading the glyph data.
6. It generates the missing pk fonts, if necessary, from the mf files, or from PostScript Type 1 fonts.
7. It can maintain the size of the dynamic font directories below the upper bound specified by the system administrator.

Applicable Fonts

Four different kinds of fonts can be freely intermingled while typesetting:

1. pk fonts;
2. scalable fonts resident in the printer;
3. scalable fonts for emulating typesetters; and
4. scalable fonts in (encrypted) PostScript Type 1 format.

The *Driver* determines which font data among the actually implemented ones in the host computer or in the printer are to be used for each font specified in the DVI file, by looking at the font table which describes the corresponding relationship between them.

pk fonts. There are three categories of pk files, reflecting how they are generated:

1. pk fonts generated from mf file by METAFONT,
2. pk fonts created from PostScript font (typically Type 1 format font), and
3. pk fonts which have no corresponding generic data file mentioned above.

These fonts are located in three separate font subdirectories for ease of maintenance.

The standard TEX fonts, including Computer Modern, Computer Concrete, and AMS Euler, belong to the first category. The *Driver* presumes that the pk file of this category is always available. If found missing, the *Driver* immediately trips to the automatic font generation procedure to be described later.

The pk fonts of the second category are those generated mostly from the Type 1 format PostScript fonts. If the *Driver* discovers that the needed pk font of this category is missing while the corresponding Type 1 format file exists, the automatic font generation process is activated.

The pk font of the third category does not have its associated generic file. Some of the Japanese character fonts in wide use belong to this category.

The pk glyph files in the directories corresponding to the first two categories may be erased, if necessary, during the autonomous management of the disk space, whereas the files in the third category should never be erased.

Resident scalable fonts. The printer has some number of fonts as resident scalable fonts. When two optional font cards are mounted to the LIPS version of printer, for example, the *Driver* utilizes at least nine scalable fonts simultaneously, namely Dutch, Swiss, Avant Garde, Century Schoolbook, Bookman, Zapf, together with three standard Japanese typefaces, that is, Mincho, Square Gothic, and Round Gothic.

Scalable fonts for emulating typesetters. The font glyph data described by Bézier curves in conformance with a subset of PostScript language constructs are also available. They are derived from the proprietary fonts originally developed in Ikarus format for some of commercial typesetters. (As such they are not in PostScript font format.) Although this type of font is not necessarily in wide circulation, there are occasions when one wishes to simulate the typesetter with a local printer. Publishers and authors might be able to enjoy improved mutual communications.

Outline font data are sent to the printer after translating into the control language of the printer.

PostScript Type 1 format fonts. The fonts in this category can be used in two different ways:

1. The first method converts the data into pk format in advance, as described before, and then uses it as the pk font.
2. The second method converts the content of the Type 1 format data into a sequence of plain PostScript commands by means of the decryption algorithm as specified by Adobe Systems, Inc., and then sends it to the printer just as described for the plain PostScript fonts.

The advantage of the first method is its printing speed. It is best suited for routine use or for printing a document of significant volume. Since *xdvi* can display any pk font on the X-window screen, this scheme permits brisk and efficient previewing as well. In contrast, the second method sports the advantages of a scalable font. For example, gray scale inking is possible only when the second method is used.

Gray Scale Inking

Whereas in standard documents there is no particular need for printing glyphs with anything other than solid black ink, the introduction of gray scale inking to TeX offers the following merits:

1. It adds slightly more artistic flavor to typesetting, while balancing optical reflectivity of the entire page by making a glyph of a significantly large size font look milder.
2. Controlled level of emphasis may be given to each character of a string.

Glyphs of scalable fonts may be filled with any of the hundred grades of gray scale, ranging from pure white to solid black, with or without superimposing its contour line.

Since the inking on the glyph is treated logically as opaque, a glyph of lighter inking may be superimposed on top of a darker background.

Filling with Patterns

The printer under consideration is equipped with a primitive command which fills the local region enclosed by a closed circuit such as one element of a glyph. The filling is not limited to a homogeneous gray. Any of the 64 textural filling patterns the printer can generate, in addition to the 100 homogeneous gray scales, may be equally applicable. In what follows it should be understood, therefore, that the word 'filling' refers both to gray scales and to textural patterns without any distinction.

Other Character Ornamentations

For printing instruction manuals covering subjects associated with the screens of computer terminals, it is customary to reproduce the highlighted character strings displayed in reverse video mode faithfully on a printed page. The *Driver* can perform the character black/white reversing for this purpose.

Character shading may be a milder alternative of emphasizing character strings. Any of the filling patterns may be specified for the shading.

Drop shadowing (shadow characters) may offer the effect of somewhat artistic flavor. This can be accomplished by adding an arbitrary depth and pattern of shadow to each character.

To create various kinds of forms, the *Driver* may apply shading to a rectangular area.

Insertion of Figures

The way figures are inserted into TEX documents assimilates the manner adopted by other existing DVI drivers for PostScript printers.

Any file in encapsulated PostScript format (EPSF) is qualified as a figure file. To insert a figure, a new TEX macro is called in, which requests the designated file name, and as an option the figure size as well. The *Driver* modifies, if necessary, the width and the height of the figure independently so that the size matches exactly that specified by the macro parameters. The calculation of the magnification factors is based on the data of the original figure size contained in the header portion of the figure file.

As a contingency measure, the *Driver* accepts as well plain PostScript files which are not in EPSF. Since no rescaling is performed in this case, it is the user's responsibility to adjust beforehand the figure size to the parameters specified in the macro parameters. This extra option is to safeguard dedicated TEX adherents from the confusion in discriminating between the two PostScript formats.

A bitmap pattern, such as a dumped-out copy of an X-window screen, may also be included into a document. In most cases, however, bitmap data converted to EPSF is preferred because of the auto-scaling feature mentioned above.

Automatic Font Generation

If the *Driver* discovers that any of the pk fonts specified in the DVI file are missing, it generates the missing pk font automatically, provided that (1) the corresponding mf file exists, or (2) the font table registers the font to be generated from the corresponding PostScript Type 1 font and the PostScript font file actually exists.

In the first case, the *Driver* first activates *virmf*, which is a tool of METAFONT, to generate the pk file on the spot from the mf file, and then resumes the suspended printing work by using the newly created font, in the same way as *dvips*, and also *xdvi*, do in a similar situation.

In the latter case, it creates the pk file automatically with the help of GNU's *Font Utilities*, and then comes back to the suspended job again just as it does while working with mf files.

This feature, which helps reduce the chance of failure resulting from lack of fonts, will be found effective when printing jobs are queued across the network.

Processing Speed

The speed of printing is as important as the quality of output from the utilitarian point of view in daily service. Since the printer's internal buffer memory for storing font glyphs is limited to typically 6 MB in size, the *Driver* gives higher priority to the more frequently used fonts. The selection algorithm for the font registration checks the font size, the classification of the nature of font, and the statistics on the frequency of request in the past.

The speed of printing is greatest when the *Driver* uses the fonts which have been registered in the printer memory through the previous font downloading.

Because of the sheer volume of font data, printing Japanese documents is much more critical compared with the case with alphabetic fonts. Each Japanese font consists of 7285 characters. Without an adequate policy for the selection of font registration, precious resources will be used up quickly by fonts with less urgent needs, thus resulting in serious degradation of the performance.

Since the interface of the printer in use by the host computer is limited either to Centronics format or to RS232C, time spent in transmitting the glyph data governs the overall printing speed. Therefore it is extremely time-consuming to transmit each of the glyph data of fonts in frequent use from the host computer without storing them in the printer memory.

So as to boost the printing speed, the glyph data of the resident scalable fonts may also be registered into the buffer memory. While using multiple Japanese fonts, however, there occurs occasionally a situation when the amount of font data exceeds the volume of the buffer memory. In these circumstances, use of the resident fonts without registration can be a good compromise because it eliminates at least the time loss due to the glyph data transfer.

Compatibility

To eliminate system dependencies as much as possible from TEX source files while using the newly added extended features (gray scale inking, pattern filling, and character ornamentation), a style file which contains the related macros definitions is included in the (LA)TEX source files. Under the premise that these features are universally welcome by a broad spectrum of the audience, it is hoped that TEX source files can attain a certain degree of system independence when the other printer drivers eventually incorporate their respective style files sharing the common format for these macros.

Evaluation

The *Driver* has been tested successfully with all models of laser-beam printer which were available for evaluation: Canon's B406S, A304E, A404E, and B406G. All of them are supported with the LIPS control language.

The measurement with the printer controlled by a 32-bit RISC chip at 600 dpi internal resolution (B406G) indicates that the printing speed reaches nearly the speed governed by the printer's paper handling rate for typical TEX documents without figures.

The average time needed to create a new pk font from a PostScript Type 1 font is about 20 seconds with a typical workstation, Sony's NWS-3860 (CPU: R3000 at 20MHz). Incidentally, it took slightly less than 19 hours to generate, in a single batch, seven standard sizes (10 pt and its assorted six magnifications up to \magstep5) of the pk fonts each for the entire 500 typefaces contained in *URW TypeWorks* compact disc. The generated 3500 pk files occupy 31 MB of disk space.

A full-page sample output is reproduced in the Appendix. To demonstrate the *Driver*'s ability of font handling and character ornamentation, a somewhat chaotic melange of fonts in various formats and sizes have been selected. It is hoped that the base font size as 20pt, or in part \magstep3 of 10pt, is large enough to make the finer detail of decorative features of the original 600 dpi output reproducible after printing.

It is emphasized that no PostScript processing is involved in generating this sample page. It simply generates the intrinsic command codes of the printer.

Implementation and Availability

The *Driver* is written entirely in C language. The total amount of source files of the current version is 190 Kb, or 8570 lines, excluding files associated with fonts data. It can be compiled using GNU's *gcc* compiler.

The standard TEX environment of Unix is the minimum prerequisite for porting the *Driver*. With this basic environment, printer-resident scalable fonts with ornamenting capability, together with pk fonts, can be used. In addition, GNU's *Ghostscript* has to be available in order to include figures in EPSF format. Likewise, GNU's *Font Utilities* is needed to deal with PostScript Type 1 fonts.

A revised version, which is currently being developed with improved portability and ease of maintenance in mind, will facilitate easier implementation on various Unix platforms, including 4.3BSD, OSF/1 and SunOS. This version, after subsequent field tests, will be available as free software by accessing, through anonymous ftp, the directory

`/ftp_services/TeX_driver`

at the address

`i1nws1.fuis.fukui-u.ac.jp (133.7.35.53)`.

Conclusions

With the DVI driver for non-PostScript printers, the following assets have been realized:

A variety of fonts can be incorporated, including pk fonts, resident scalable fonts of the printer, and PostScript Type 1 fonts. As a part of performance evaluation, the *Driver* has been put into daily service incorporating, among others, 3500 pk fonts (500 distinct typefaces, each in 7 sizes) generated from a single CD-ROM of PostScript Type 1 format fonts.

Æsthetic features and visual appeal, essential in certain kinds of documents, can be realized conveniently with the knowledge of TEX alone. Character ornamentation, such as filling glyphs with a pattern, drop-shadowing, black/white reversing, and outlining, can be achieved without recourse to other technology, such as PostScript.

The insertion of figures by means of an encapsulated PostScript format file has been realized. The identical format to that used in PostScript printers has been acknowledged for compatibility.

Acknowledgements

The author acknowledges Ken-ichi Sugimoto, Atsushi Takagi, and Tetsuto Sakai for their effort in coding most of the software as part of their graduate study. He thanks Canon Corp. for offering printers for field evaluation. *URW TypeWorks* compact disc, used as the major source of PostScript Type 1 fonts, was supplied by Seaside Software Inc., Chigasaki, Japan. Special scalable fonts for emulating typesetters were supplied by Heidelberg PMT Co., Ltd., and also by Dainippon Printing Co., Ltd., both in Japan.

The *Driver* is indebted to various existing tools in one phase of its operation or another: *Ghostscript* and its adaptation to LIPS printers, *dvips*, *dvi2ps*,

Font Utilities, and METAFONT. Appreciations to their respective authors should be due in letting their tools be widely available.

Bibliography

Adobe Systems Inc. *Adobe Type 1 Font Format*, Addison-Wesley, 1990

Berry, Karl, and Kathryn A. Hargreaves. "Font utilities" version 0.6, 1992

Cooper, Eric, Bob Scheifler, Paal Kvamme, Håvard Eidnes, Mark Eichin, Paul Vojta, Jeffrey Lee, Donald Richardson, *et al.* "xdvi on-line manual", 1992.

Deutsch, L. Peter. "Ghostscript 2.6.1", 1993.

Knuth, Donald E. *The METAFONTbook*, Addison-Wesley, 1986.

Rokicki, Tomas. "DVIPS: A TEX Driver".

Sagiya, Yoshiteru, Hiroshi Ishii, Hajime Kobayashi, Ryouichi Kurasawa, and Hisato Hamano. *Japanese TEX Technical Book I*, ASCII Corp., 1990.

Sample Output

Selection of solid black, 50% gray scale, and outlined pure white inking at 30 pt: **Cheltenham Ultra Italic**.

Outline may be superimposed to the gray

TEX

glyph. The logo TEX uses one of 64 filling patterns for 50 pt **Bauhaus**. *[Font: Avant Garde]*

Character B/W Reversing is indispensable in expressing the inverted video display on the CRT screen.

Character Shading works for mild emphasis or printing forms.

Line Widths of Outline specified are 2pt, 4pt, and 6pt. *[Font: Century Schoolbook]*

Some shadowing examples: **Shadow + Patternfill** , **Shadow + Shading** , **Shadow + Outline** , 15% Right-Below , and 6% Down .

Character ornamentation shown above can be applied to the scalable fonts, irrespective of the font size. *[Font: Bookman]*

*Exotic fonts like script styles go just as well with TEX: Freestyle, Aritus, Englische Schreibschrift, Commercial, Phyllis Initials, Park Avenue, Murray Hill, **Brush Script**, to name a few. Japanese fonts, like* 内蔵 明朝体 *or* 輪郭を描く*, in arbitrary size can be used and ornamented.*

Typesetting Catalan Texts with TEX

Gabriel Valiente Feruglio
Universitat de les Illes Balears
Departament de Ciències Matemàtiques i Informàtica
E-07071 Palma de Mallorca (Spain)
Internet: dmigva0@ps.uib.es

Robert Fuster
Universitat Politècnica de València
Departament de Matemàtica Aplicada
Camí de Vera, 14. E-46071 València (Spain)
Internet: mat5rfc@cci.upv.es

Abstract

As with other non-American English languages, typesetting Catalan texts imposes some special requirements on TEX. These include a particular set of hyphenation patterns and support for a special ligature: unlike other Romanic languages, Catalan incorporates the middle point in the l·l digraph. Hyphenation rules for Catalan are reviewed in this paper, after a short introduction to hyphenation by TEX. A minimal set of hyphenation patterns covering all Catalan accents and diacritics is also presented. A discussion about the l·l ligature concludes the paper. This work represents a first step towards the Catalan TLP (TEX Language Package), under development within the TWGMLC (Technical Working Group on Multiple Language Coordination), where the first author is chairing the Catalan linguistic subgroup.

Resum

Així com en altres llengües, la composició de textos escrits en català demana requeriments especials al TEX. Aquests inclouen un conjunt particular de patrons de guionat, així com suport per a un lligam especial ja que, a diferència d'altres llengües romàniques, el català incorpora el punt volat al dígraf l·l. En aquest paper es fa una introducció al guionat amb TEX i es revisen les regles de guionat per al català. Tanmateix, es presenta un conjunt mínim de patrons de guionat que cobreix tots els accents i marques diacrítiques del català. El paper acaba amb una discussió sobre el lligam l·l. Aquest treball representa un primer pas cap al TLP (Paquet de Llengua TEX) català que s'està desenvolupant dins el TWGMLC (Grup Tècnic de Treball sobre Coordinació de Múltiples Llengües), on el primer autor presideix el subgrup lingüístic català.

Hyphenation by TEX

Background on hyphenation by TEX is first presented, following the ninth edition of *The TEXbook* (Knuth, 1990) and the exposition in Haralambous (*TUGboat*, 1990). The actual hyphenation algorithm used by TEX is due to Liang (1983).

When TEX creates a format file like `plain.fmt`, `lplain.fmt` or `amsplain.fmt`, it reads information from a file called `hyphen.tex` (or `**hyphen.tex`, where `**` is a two-letter language code[1] (see Haralambous, *TEX and TUG NEWS*, 1992)) that contains the *hyphenation patterns* for a specific language. Using TEX3+, a format file can include more than one (up to 256) sets of patterns and, so, INITEX produces multilingual versions of TEX. In this case, language-switching mechanisms like those of the Babel system by Johannes Braams allow TEX to typeset every language according to its own rules. A syntax

[1] In the Catalan case, the name of this file will be `cahyphen.tex`.

for language-switching commands has not yet been standarized, but it is expected to be something like

```
\language{catalan}{...Catalan text...}
```

for short inserts and

```
\begin{language}{catalan}
...Catalan text...
\end{language}
```

for longer inserts.

Hyphenation patterns are clusters consisting of letters separated by digits, like x1y2z (more exactly, a pattern has the form

number/letter/number/letter/.../number

like 0x1y2z0, but the number 0 can be suppressed), meaning that:

- If the set of patterns is empty, no hyphenation takes place.
- If there is a pattern x1y, then hyphenation "x-y" will be possible in every occurrence of the cluster "xy". If the pattern is x1yzw, then the sequence of letters "xy" will be hyphenated only when followed by "zw".
- If there is a pattern x1y and a pattern x2yabc then the sequence "xy" will be hyphenated, as long as it is not followed by "abc". The digit 2 indicates therefore an exception to the rule "separate x and y" expressed by the digit 1.
- The same holds for greater numbers. Patterns with number 3 will be exceptions to patterns with number 2, and so on: odd numbers allow and even numbers disallow hyphenation, and the maximum decides.
- A dot in front of (or behind) a pattern, such as .x1y or xy2z. specifies that the pattern is valid only at the beginning (or at the end) of a word.

In this context, a *letter* is a character of category 11 or 12 whose \lccode is nonzero. Because, for almost all Latin-alphabet languages, some diacriticized characters are letters for which we need a mechanism, including these special characters as letters. Using TeX3+, which allows 8-bit input, this problem disappears.

Despite the existence of some fundamental rules, hyphenation of a particular language can be very complicated. There are two methods to handle this complexity: hidden mechanisms of hyphenation can be investigated and patterns made to correspond to the analytical steps of manual hyphenation, or patterns can be induced from a sufficiently representative set of already hyphenated words, using inductive inference tools tailored to this particular problem such as PATGEN.

The choice of method depends on the nature of the language and on the size of the available set of hyphenated words. Although in theory such a pattern generator would produce an *exhaustive* set of patterns from a file containing *all* words of a particular language in hyphenated form, it is more probable to have partial sets of hyphenated words, and the pattern generator will only produce more or less accurate approximations.

The authors have chosen the first method for Catalan. Besides hyphenation patterns, the effort resulted in more systematic and exhaustive rules for Catalan hyphenation than those found in grammar textbooks.

Catalan Hyphenation Rules and Patterns

Modern Catalan normative grammar was established by Pompeu Fabra and ratified by the *Institut d'Estudis Catalans* (Catalan Studies Institute) in 1917. Orthography (and in particular syllabification and hyphenation rules) can be found in many texts: Bruguera (1990), Fabra (1927), Mira (1974), Pitarch (1983), Salvador (1974), and many others. The official normative dictionary is *Diccionari general de la llengua catalana* (Fabra, 1974) and *Diccionari ortogràfic i de pronúncia* (Bruguera, 1990) is a hyphenation dictionary. A very interesting study of some difficulties in the Catalan orthography can be found in Solà (1990). Some of our observations on Spanish, Italian or French hyphenation were suggested by the preceding references, but also by Lázaro (1973) and Beccari (1992).

Catalan, like other Romanic languages, bases its hyphenation rules on the syllabic structure of words. This structure, as far as Catalan is concerned, is closely related to Spanish, Portuguese or Italian. But there exist a number of differences: for example, the Catalan word *València* has four syllables and the Spanish *Valencia* has only three.

Of course, the Catalan alphabet follows the standard Latin alphabet. The letters k and w never appear (except in foreign words), the letter y is only used to form the digraph ny and letter q only appears followed by letter u.

In general a Catalan word has as many syllables as it has vowels, either separated by consonants or contiguous but not forming diphthongs. In fact, a Catalan word has *exactly* as many syllables as it has vowels, but in some special cases, letters i, u are not vowels (Catalan vowels are a, e, i, o and u). Word stress, however, determines how a Catalan word breaks up into syllables and, in some polysyllabic words, is expressed by an accent on the vowel of the stressed syllable. In this way, accents in Catalan are used in nearly the same way as in Spanish. Also as in Spanish, the accents perform another diacritic function (to distinguish some homophones, as *dona* = woman and *dóna* = he/she gives). However, the kind of accent, grave (`) or acute (´), marks the difference between open and closed vowels, as in French or Italian: so, all accented vowels are à, è, é, í, ò, ó and ú. The diaeresis (¨), over i or u, splits a diphthong or causes the letter u to be pronounced when g or q precede it.

The cedilla under the letter c (ç) and the apostrophe (') are usual in Catalan, with the same use as in French. Virtually all European languages have their own particularities: Catalan has the special construction l·l.

Syllabification. Basic rules for word division into syllables include the following (v, $v_n, n \geq 1$ will be vowels and c, $c_n, n \geq 1$ consonants).

1. A single consonant between two vowels forms a syllable with the vowel that follows it: v_1-cv_2. Actually it suffices to consider patterns of the form -cv, because if another consonant (instead of the first vowel) precedes c the pattern would also be c_1-c_2v (see rules 2, 3 and 5 below). The necessary patterns will be:

```
1ba 1be 1bi 1bo 1bu
1ca 1ce 1ci 1co 1cu
1ça         1ço 1çu
1da 1de 1di 1do 1du
1fa 1fe 1fi 1fo 1fu
1ga 1ge 1gi 1go 1gu
1ha 1he 1hi 1ho 1hu
1ja 1je 1ji 1jo 1ju
1la 1le 1li 1lo 1lu
1ma 1me 1mi 1mo 1mu
1na 1ne 1ni 1no 1nu
1pa 1pe 1pi 1po 1pu
1ra 1re 1ri 1ro 1ru
1sa 1se 1si 1so 1su
1ta 1te 1ti 1to 1tu
1va 1ve 1vi 1vo 1vu
1xa 1xe 1xi 1xo 1xu
1za 1ze 1zi 1zo 1zu

1bà 1bè 1bé 1bí 1bò 1bó 1bú
1cà 1cè 1cé 1cí 1cò 1có 1cú
1çà             1çò 1çó 1çú
1dà 1dè 1dé 1dí 1dò 1dó 1dú
1fà 1fè 1fé 1fí 1fò 1fó 1fú
1gà 1gè 1gé 1gí 1gò 1gó 1gú
1hà 1hè 1hé 1hí 1hò 1hó 1hú
1jà 1jè 1jé 1jí 1jò 1jó 1jú
1là 1lè 1lé 1lí 1lò 1ló 1lú
1mà 1mè 1mé 1mí 1mò 1mó 1mú
1nà 1nè 1né 1ní 1nò 1nó 1nú
1pà 1pè 1pé 1pí 1pò 1pó 1pú
1rà 1rè 1ré 1rí 1rò 1ró 1rú
1sà 1sè 1sé 1sí 1sò 1só 1sú
1tà 1tè 1té 1tí 1tò 1tó 1tú
1và 1vè 1vé 1ví 1vò 1vó 1vú
1xà 1xè 1xé 1xí 1xò 1xó 1xú
1zà 1zè 1zé 1zí 1zò 1zó 1zú
```

2. Of two consonants standing between two vowels, the first forms a syllable with the preceding vowel and the second forms a syllable with the vowel that follows it: v_1c_1-c_2v_2. Because the preceding patterns allow this break, we do not need special patterns for this rule. But one exception to this rule is that the liquid consonants, l and r, when preceded by certain consonants, form a syllable with this consonant and the vowel that follows. Another exception is that there are some special combinations, called *digraphs*, that represent only one phoneme or a geminated one. The complete list is: ig, ix, ll, l·l, ny, rr, ss, tg, tj, tl, tll, tx, tz. The digraph ig only occurs at the end of a word (and in plural form, igs).

   The two following rules exactly define these exceptions.
3. The combinations c-l and c-r that cannot be hyphenated are bl, cl, fl, gl, pl, br, cr, dr, fr, gr and pr. The necessary patterns will be:

```
1b2l 1c2l      1f2l 1g2l 1p2l
1b2r 1c2r 1d2r 1f2r 1g2r 1p2r 1t2r
```

   The combination vr is another one that cannot be hyphenated, but it appears only in a few toponymies.
4. The digraphs ll and ny are not split (following an etymological criterium). The pattern

```
l2l
```

   voids the effect of the first rule. No analogous pattern is necessary for ny. In fact, ny and ll correspond to single consonant sounds and therefore rule 1 above applies to them as well. The necessary patterns will be:

```
1lla 1lle 1lli 1llo 1llu
1llà 1llè 1llé 1llí 1llò 1lló 1llú
1nya 1nye 1nyi 1nyo 1nyu
1nyà 1nyè 1nyé 1nyí 1nyò 1nyó 1nyú
```

   All other digraphs can be split. The l·l ligature is also a digraph and can be divided, replacing the middle dot with a hyphen. Hyphenation of the l·l ligature is discussed in the next section.
5. Of three or more consecutive consonants followed by a vowel, the last consonant forms a syllable with that vowel: c_1c_2-c_3v, et cetera, unless the last two consonants belong to those in the two rules above. No additional patterns are necessary for this rule.
6. Compound words with one of the following prefixes

an, con, des, en, ex, in, sub, trans

are divided according to components and therefore often constitute exceptions to the previous rules. These differ from prefix to prefix and present an evident problem: it is impossible, unless you make an exhaustive classification by scanning a dictionary, to determine if a certain combination is or is not a prefix.[2] For example, you must hyphenate *in-a-pe-tèn-ci-a* (inappetence) but *e-nò-leg* (an expert in wine) instead of *en-ò-leg*. For instance, using Bruguera (1990), we find the following patterns for .ex:

```
.e2x1a .e2x1à
.e3x2ag .e3x2am .e3x2àm
.e2x1on .e2x1or .e2x1osm
.e3x2orc .e3x2ord
.e2x1u1c
```

In all words starting with trans — except in transit and its derivatives — trans is a prefix. Then, the corresponding patterns will be

```
.tran2s1 .tran3s2i
```

Because these prefixes are very frequent in practice and — specially in technical languages — frequently used to create new words, this is a dangerous solution. Another possible solution — more conservative, but completely secure — consists of inhibiting the splitting of such a group whenever it is present at the beginning of a word (except in the case of trans, because it is a very long prefix):

```
.a2n .co2n .de2s .e2n .e2x
.i2n .su2b .tran2s .tran3s2i
```

To choose between these two options is still an open question.

7. Personal pronouns *nosaltres* (we) and *vosaltres* (you) are etymologically composed words. They must, therefore, be hyphenated *nos-al-tres, vos-al-tres*.[3]. The necessary patterns are:

```
.no2s1a1 .vo2s1a
```

Exceptions to the syllabification rules above are certain groups of vowels where i or u are not really vowels. The next sections explain these exceptions.

Descending Diphthongs. When a vowel is followed by an unstressed i or *u*, this second letter is a semivowel and forms a syllable with the preceding vowel. These diphthongs are ai, ei, oi, ui, au, eu, iu, ou and uu.

All other combinations of two vowels are divided. The necessary patterns will be:

```
a1a a1à a1e a1è a1é     a1í a1o a1ò a1ó a1ú
e1a e1à e1e e1è e1é     e1í e1o e1ò e1ó e1ú
i1a i1à i1e i1è i1é i1i i1í i1o i1ò i1ó i1ú
o1a o1à o1e o1è o1é     o1í o1o o1ò o1ó o1ú
u1a u1à u1e u1è u1é     u1í u1o u1ò u1ó u1ú

à1a è1a é1a í1a ò1a ó1a ú1a
à1e è1e é1e í1e ò1e ó1e ú1e
            í1i
à1o è1o é1o í1o ò1o ó1o ú1o
```

Ascending Diphthongs and silent u. When the letters g or q come before the vowel u and another vowel, then either the u is not pronounced or the two vowels compose an ascending diphthong (and the u is a semiconsonant).[4] In both cases, the three letters belong to the same syllable and the combination cannot be hyphenated. Then, we need to make void some of the preceding patterns. For the g the patterns will be:

```
gu2a gu2e gu2o
gu2à gu2è gu2é gu2í gu2ò gu2ó
```

The letter q is used only in this context and always starting a syllable. Then the only necessary pattern is

```
1qu2
```

Triphthongs. Yet another exception to the syllabification rules above is a group of three vowels (actually, a semiconsonant/vowel/semivowel combination) that together constitute a single syllable. The only triphthong in Catalan is uai after g or

[2] PATGEN allows an *adjustment of an existing set of patterns*; it will read both a set of already existing patterns and a collection of hyphenated words, and will create a new set of patterns. This method can be used as a combination of the analytical and the raw PATGEN methods. For example, one could extract *all* words starting with one of the prefixes an, con, des, en, ex, in, sub and trans, from a dictionary, and run PATGEN on these and on the existing patterns. The test function of PATGEN will immediately evaluate if the new set of patterns is more powerful.

[3] In 1959 the *Academia Española de la Lengua* (Spanish Language Academy) revoked a similar prescriptive rule. So, in Spanish you can hyphenate *nos-otros* or *no-sotros* This applies also to the Spanish hyphenation of the prefix des.

[4] These are the only cases of ascending diphthongs in Catalan. It differs from Spanish and Italian: in these two languages all combinations of i or u with a vowel are diphthongs.

q, but no special patterns are necessary because the preceding patterns gu2a 1qu2 apply and the combination ai is not hyphenated.

Letter i or u as consonant. Unstressed i before a, e or o, however, becomes a consonant when situated at the beginning of a word (even when preceded by h), except in ió and its derivatives. The necessary patterns will be:

```
.i2a .i2à .i2e .i2è .i2é .i2o .i2ò
.hi2a .hi2e .hi2o
.hi2à .hi2è .hi2é .hi2ò .hi2ó
.i3on
```

Unstressed i or u standing between vowels are consonants and form a syllable with the vowel that follows it. The necessary patterns will be:

```
a1i2a a1i2e a1i2i a1i2o a1iu
e1i2a e1i2e e1i2i e1i2o e1iu
ii2a  ii2e        ii2o
o1i2a o1i2e o1i2i o1i2o o1iu
u1i2a u1i2e u1i2i u1i2o u1iu

a1u2a a1u2e a1ui a1u2o a1uu
e1u2a e1u2e e1ui e1u2o e1uu
i1u2a i1u2e i1ui i1u2o i1uu
o1u2a o1u2e o1ui o1u2o o1uu
u1u2a u1u2e u1ui u1u2o

à1i2a à1i2e à1i2i à1i2o à1iu
à1u2a à1u2e à1ui  à1u2o à1uu
è1i2a è1i2e è1i2i è1i2o è1iu
è1u2a è1u2e è1ui  è1u2o è1uu
ò1i2a ò1i2e ò1i2i ò1i2o ò1iu
ò1u2a ò1u2e ò1ui  ò1u2o ò1uu
é1i2a é1i2e é1i2i é1i2o é1iu
é1u2a é1u2e é1ui  é1u2o é1uu
í1i2a í1i2e       í1i2o
í1u2a í1u2e í1ui  í1u2o í1uu
ó1i2a ó1i2e ó1i2i ó1i2o ó1iu
ó1u2a ó1u2e ó1ui  ó1u2o ó1uu
ú1i2a ú1i2e ú1i2i ú1i2o ú1iu
ú1u2a ú1u2e ú1ui  ú1u2o

a1i2à a1i2è a1i2ò a1u2à a1u2è a1u2ò
e1i2à e1i2è e1i2ò e1u2à e1u2è e1u2ò
ii2à  ii2è  ii2ò  i1u2à i1u2è i1u2ò
o1i2à o1i2è o1i2ò o1u2à o1u2è o1u2ò
u1i2à u1i2è u1i2ò u1u2à u1u2è u1u2ò
a1i2é a1i2í a1i2ó a1i2ú
a1u2é a1u2í a1u2ó a1u2ú
e1i2é e1i2í e1i2ó e1i2ú
e1u2é e1u2í e1u2ó e1u2ú
o1i2é o1i2í o1i2ó o1i2ú
o1u2é o1u2í o1u2ó o1u2ú
```

Diaereses. In Catalan the diaeresis is used in two different contexts: first, if an i or u — following a vowel or between two vowels — is a real vowel (and in consequence does not belong to the same syllable). But, second, in the combinations qüe, güe, qüi, güi it indicates that the u is pronounced (forming a diphthong with the following vowel).

The corresponding patterns are:

```
a1ï e1ï i1ï o1ï u1ï a1ü e1ü i1ü o1ü u1ü
ï1a ï1e ï1i ï1o ï1u ü1a ü1e ü1i ü1o ü1u
1gü2 1qü2
ü3ï
```

The last pattern applies to a very special case: in *argüïen* and other related words appear two consecutive diaereses (Valor (1983), p. 20).

Breaks. Catalan words may be broken into syllables containing just one letter. Actually, only vowels can form a syllable on their own, but some learned words or words of foreign origin, like *psicòleg* or *show* start with a pair of consonants: the possible combinations are gn, mn, pn, ps, sc, sh, sl, sm, sn, sp, st, ts; the only occurrence of a digraph beginning a word is in the word *txec* and its derivatives (as *txecoslovac*). Then, the following patterns are necessary in order to make void the effect of the first rule and to prevent separating single consonants at the beginning of words:

```
.g2 .m2 .p2 .s2 .t2
```

Also combinations like cl and br can start a word, but then rule 3 applies and no special patterns are required.

Finally, to prevent hyphenation of an apostrophe, we only need the pattern

```
'2h
```

Now we have a complete set of hyphenation patterns, even if the parameters \lefthyphenmin and \righthyphenmin are set to 1. Regarding this question, we suggest the values

```
\lefthyphenmin=1 \righthyphenmin=3
```

because long ending syllables are frequent in Catalan words and then, with the default values, very frequent words like *aquests* (the plural masculine demonstrative) must not be hyphenated. So, the macros involved in the Catalan language LaTeX environment should include:

```
\language=2 % or the appropriate value
\lccode`\'=`\'
\nonfrenchspacing
\lefthyphenmin=1
\righthyphenmin=3
```

The ŀl Ligature

All Catalan characters belong to the ISO 8859-1 coding scheme, known as ISO LATIN-1, with only one exception. Double *ll* also exhibits a geminated form, *ŀl*. Let us take a look at its etymology.

While some Romanic languages preserve the phonetic distinction between |λ| and |*ll*|, in particular French, Italian and Catalan, it is only in

Catalan where this phonetic distinction finds a corresponding orthographic distinction. For instance, Latin INTELLIGENTIA derives into French *intelligence* and Italian *intelligenza*, while Latin SELLA derives into French *selle* and Italian *sella*. Then these languages use the same orthography for two different phonemes.

Modern Catalan uses **ll** for phoneme $|\lambda|$ and **l·l** for phoneme $|ll|$. Then Latin INTELLIGENTIA derives into Catalan *intel·ligència* and Latin SELLA derives into Catalan *sella*.

This correspondence between phonetics and orthography is a debt to the normalization process to which Catalan has been subject to, where Pompeu Fabra (1984) has played a fundamental role. Early grammar texts, however, use **ł** for **ll** and **ll** for **l·l** (Fabra, 1912). See Fabra (1983, 1984) and Segarra (*Història de l'ortografia catalana*, 1985) for more details on these orthographic distinctions[6].

The l·l ligature and DC fonts. This section is a revised excerpt from discussions held between Gabriel Valiente Feruglio and Yannis Haralambous during 1992, while contributing to Haralambous' efforts in incorporating national requirements from different countries into the design of DC fonts.

Q. Is it necessary or facultative (like the fi ligature)?

A. It is mandatory.

Q. Is there also an "ll" without dot?

A. Yes, there is. The "ll" without dot corresponds to a palatal sound, while the "l·l" with middle dot corresponds to the "gemination" or duplication of the "l" sound.

Q. What is its uppercase counterpart?

A. The l·l ligature cannot appear at the beginning of a word, only joining two syllables. Therefore, the only way in which the l·l must be shown in uppercase is when the whole word is in uppercase, and in such a case both L's are capitalized, as the word INTEL·LIGENCIA shows.

Q. How do you create it using TEX and/or other word processors?

A. Detailed definitions for TEX are given and discussed in the next section. Many WYSIWYG word processors actually support the l·l ligature, that is obtained by joining two characters: an *l* with middle dot (l·) and another *l*. When hyphenation takes place, the *l·* gets replaced by a normal *l*.

Q. Can it be hyphenated?

A. The function of l·l can be seen as that of joining two syllables, one ending in "l" and the other beginning with "l". Therefore, it can be hyphenated, and the right hyphenation is "l-" and "l". For instance, the word *intel·ligència* would be hyphenated as: `in-tel-li-gèn-ci-a`. It is therefore a ligature instead of a single character. This justifies the lack of an l·l character in DC fonts, although a middle dot other than TEX's centered dot `$\cdot$` could also be useful, besides Catalan, for other languages as well.

Q. What is its alphabetical order?

A. It does not appear in the alphabetical order, because it has no extra sound, just the mere duplication of the "l" sound. [Comment of R. Fuster: Colomer (1989), a Catalan-English/English-Catalan dictionary, and Bruguera (1990) arrange cella before cel·la. But Fabra (1974), Ferrer (1973) and Romeu et al. (1985) give this order: cel·la, cella.]

Q. What are the local encoding schemes used? Are there Catalan keyboards with ·, l· or l·l support?

A. A centered dot appears in ISO 8859-3 as character 0xB7, and the character combinations LATIN CAPITAL LETTER L WITH MIDDLE DOT and LATIN SMALL LETTER L WITH MIDDLE DOT appear in positions 0x3F and 0x40 of row 01 (EXTENDED LATIN A) of ISO IEC DIS 10646-1.2. Besides these ISO codes for middle dot, character sets for Personal Computers happen to include a special "l·" character, often in the Danish or Norwegian code pages.

Q. Can it appear in ligatures, like fl·l or ffl·l ?

A. No, it cannot. For morphological reasons l·l has to be preceded and followed by vowel sounds.

Q. Are there special spacing rules? Is the dot special?

A. Yes, the l's are closer to the dot than other letters, and the dot is a normal dot but raised approximately half the height of a vowel from the baseline for lowercase and three times that height for uppercase[7].

Q. When did this letter appear in Catalan printing?

[6] The two last paragraphs demonstrate the hyphenation of the l·l ligature, which is discussed in detail in the next section.

[7] More reasonable spacing can be achieved by raising the dot exactly the height of a lowercase vowel, and this is precisely what has been coded in the macro for the l·l ligature presented below. Thanks to Marek Ryćko and Bogusław Jackowski for their comments on that particular spacing convention during the 1993 TEX Users Group Annual Meeting.

A. Although it was Pompeu Fabra who always supported the idea of an orthographic distinction in correspondence with the phonetic distinction between $|\lambda|$ and $|ll|$, his approach consisted of leaving *ll* for $|ll|$ and looking for a new symbol for $|\lambda|$. The actual ligature *l·l* is due to Mossèn Alcover in his amendment to the fourth writing of the *Normes Ortogràfiques* (Orthographic Norms) by the *Institut d'Estudis Catalans* (Catalan Studies Institute) (Segarra, 1985). The *l·l* ligature appeared therefore in Catalan printing for the first time in 1913 in *Normes Ortogràfiques.*

Choosing a macro for the l·l ligature. When it comes to choosing the best character sequence for the TEX macro producing the l·l ligature we realize that perhaps we Catalan TEX users have arrived too late, because most short combinations already have a definition in plain TEX. Among the interesting ones are `\l` and `\L`, assigned to Polish letters ł and Ł and `\ll`, assigned to the "much less than" relation $\ll$, whereas `\LL` is undefined in plain TEX.

It must be noted, however, that $\ll$ only occurs in math mode, while the l·l ligature is not supposed to be typed in math mode. We have therefore chosen `\ll` and `\LL` as character sequences for the macro definition producing the l·l ligature, and have included a test for math mode in the definition in order to restore the original $\ll$ relation when in math mode for lowercase `\ll`, as explained in the next section.

The macro name `\ll` is submitted to the TWGMLC for standarization.

Typesetting the l·l ligature. No normative exists for typesetting the *l·l* ligature and therefore quite different kernings between the middle dot and the two consonants can be found in modern Catalan writings. The definitions

```
\newskip\zzz
\def\allowhyphens{\nobreak\hskip\zzz}
\def\ll{\allowhyphens%
  \discretionary{l-}{l}{l\hbox{$\cdot$}l}%
  \allowhyphens}
\def\LL{\allowhyphens%
  \discretionary{L-}{L}{L\hbox{$\cdot$}L}%
  \allowhyphens}
```

constitute a good starting point because, besides achieving an easy-to-read spacing, such as in il·lusió and IL·LUSIÓ, they produce the right hyphenation. Middle dot is lost and l·l is hyphenated `l-l`.

Explicit kerning can be added between middle dot and the two consonants. Because kern is font-dependent, some character height, width, and depth values for the actual font in use are taken into account in the following definitions in order to set appropriate kerning.

```
\newskip\zzz
\def\allowhyphens{\nobreak\hskip\zzz}
\newdimen\leftkern
\newdimen\rightkern
\newdimen\raisedim

\def\ll{\relax\ifmmode \mathchar"321C
  \else
    \leftkern=0pt\rightkern=0pt%
    \raisedim=0pt%
    \setbox0\hbox{l}%
    \setbox1\hbox{l\/}%
    \setbox2\hbox{x}%
    \setbox3\hbox{.}%
    \advance\raisedim by -\ht3%
    \divide\raisedim by 2%
    \advance\raisedim by \ht2%
    \ifnum\fam=7 \else
      \leftkern=-\wd0
      \divide\leftkern by 4%
      \advance\leftkern by \wd1
      \advance\leftkern by -\wd0
      \rightkern=-\wd0
      \divide\rightkern by 4%
      \advance\rightkern by -\wd1
      \advance\rightkern by \wd0
    \fi
    \allowhyphens\discretionary{l-}{l}%
    {\hbox{l}\kern\leftkern%
      \raise\raisedim\hbox{.}%
    \kern\rightkern\hbox{l}}\allowhyphens
  \fi}

\def\LL{\setbox0\hbox{L}%
    \leftkern=0pt\rightkern=0pt%
    \raisedim=0pt%
    \setbox1\hbox{L\/}%
    \setbox2\hbox{x}%
    \setbox3\hbox{.}%
    \advance\raisedim by -\ht3%
    \divide\raisedim by 2%
    \advance\raisedim by \ht2%
    \ifnum\fam=7 \else
      \leftkern=-\wd0
      \divide\leftkern by 8%
      \advance\leftkern by \wd1
      \advance\leftkern by -\wd0
      \rightkern=-\wd0
      \divide\rightkern by 6%
      \advance\rightkern by -\wd1
      \advance\rightkern by \wd0
    \fi
    \allowhyphens\discretionary{L-}{L}%
    {\hbox{L}\kern\leftkern%
      \raise\raisedim\hbox{.}%
    \kern\rightkern\hbox{L}}\allowhyphens
  }
\endinput
```

The definitions produce the following result:

`\rm`	l·l	intel·ligència	L·L	COL·LECCIÓ
`\it`	*l·l*	*intel·ligència*	*L·L*	*COL·LECCIÓ*
`\sl`	*l·l*	*intel·ligència*	*L·L*	*COL·LECCIÓ*
`\bf`	**l·l**	**intel·ligència**	**L·L**	**COL·LECCIÓ**
`\tt`	`l·l`	`intel·ligència`	`L·L`	`COL·LECCIÓ`

Availability

Besides the `\patterns` described in this paper, two other sets of hyphenation patterns exist for Catalan. They have been developed by Gonçal Badenes and Francina Turon (`badenes@imec.be`), and by Francesc Carmona (`franc@porthos.bio.ub.es`).

All three `cahyphen.tex` files are under beta test, and can be obtained from the respective authors. The authors have tested the version described here on a PC, using multilingual PCTEX 3.1, PCTEX 3.14 and emTEX 3.141 — using the primitive `\charsubdef` of Ferguson (1990) — and also on a Macintosh, using Euro-OzTEX and the Cork font scheme. Hopefully, a unified set of Catalan hyphenation patterns will soon be available by anonymous `ftp` from major TEX servers.

Acknowledgements

The authors wrote this joint work after two independent works submitted to *TUGboat* (Fuster) and to this conference (Valiente Feruglio). Barbara Beeton suggested this collaboration to us and the result is certainly better than the originals. Gonçal Badenes (IMEC, Leuven) and Francesc Carmona (Universitat de Barcelona) provided beta-test sets of hyphenation patterns. Joan Moratinos (Ajuntament de Palma de Mallorca) and Magdalena Ramon (Universitat de les Illes Balears) provided assistance in grammatical, linguistic and terminological issues. Joan Alegret (Universitat de les Illes Balears) provided guidance on etymological issues. Luz Gil and Eddy Turney (Universitat Politècnica de València) revised our English. Special thanks to all of them.

Bibliography

Beccari, C. "Computer Aided Hyphenation for Italian and Modern Latin". *TUGboat* **13**(1), 1992, pp. 23-33.

Bruguera i Talleda, J. *Diccionari ortogràfic i de pronúncia.* Enciclopèdia Catalana. Barcelona, 1990.

Colomer, J. *Nou diccionari anglès-català català-anglès.* Pòrtic, Barcelona, 1989.

Fabra, P. *Gramática de la lengua catalana.* Tipografia de "L'Avenç", Barcelona, 1912.

Fabra, P. *Ortografia catalana.* Barcino, Barcelona, 1927.

Fabra, P. *Diccionari general de la llengua catalana.* EDHASA, Barcelona, 1974.

Fabra, P. *La llengua catalana i la seva normalització.* Edicions 62 i "La Caixa", Barcelona, 1980.

Fabra, P. *Converses filològiques I.* EDHASA, Barcelona, 1983.

Fabra, P. *Converses filològiques II.* EDHASA, Barcelona, 1984.

Ferrer Pastor, F. *Vocabulari castellà-valencià valencià-castellà.* L'Estel, València, 1973.

Ferguson, M. J. *Documentation file for the use of charsublist.* June, 1990.

Haralambous, Y. 'TEX Conventions Concerning Languages". *TEX and TUG NEWS*, **1**(4), 1992, pp. 3-10.

Haralambous, Y. "Hyphenation patterns for ancient Greek and Latin". *TUGboat*, **13**(4), 1992, pp. 457-469.

Institut d'Estudis Catalans. *Normes Ortogràfiques.* Barcelona, 1913.

Knuth, D. E. *The TEXbook.* Addison-Wesley, Reading, Massachusetts, 9th printing, 1990.

Lázaro Carreter, F. *Lengua española: historia, teoría y práctica.* Anaya, Madrid, 1973.

Liang, F. M. "Word hy-phen-a-tion by com-pu-ter". Ph.D. thesis. Department of Computer Science. Stanford University, 1983.

Mira, J. F. *Som (llengua i literatura).* Edicions 3i4, València, 1974.

Pitarch, V. *Curs de llengua catalana.* Edicions 3i4, València, 1983.

Romeu, X. et al. *Diccionari Barcanova de la llengua.* Barcanova, Barcelona, 1985.

Salvador, C. *Gramàtica Valenciana (amb exercicis pràctics).* Lo Rat Penat, València, 1974.

Segarra, M. *Història de l'ortografia catalana.* Empúries, Barcelona, 1985.

Segarra, M. *Les set redaccions de les "Normes Ortogràfiques" de l'Institut d'Estudis Catalans.* In A. M. Badia i Margarit, Estudis de Llengua i Literatura Catalanes. X: Miscel·lània. Publicacions de l'Abadia de Montserrat, Barcelona, 1985.

Solà, J. *Lingüística i normativa.* Empúries, Barcelona, 1990.

Valor, E. *La flexió verbal.* Edicions 3i4, València, 1983.

The Khmer Script Tamed by the Lion (of TEX)

Yannis Haralambous
CERTAL (Centre d'Études et de Recherche sur le Traitement Automatique des Langues),
INALCO (Institut National des Langues et Civilizations Orientales), Paris, France.
Personal address: 187, rue Nationale, 59800 Lille, France
Internet: yannis@gat.citilille.fr

Abstract

This paper presents a Khmer typesetting system, based on TEX, METAFONT, and an ANSI-C filter. A 128-character of the 7-bit ASCII table for the Khmer script is proposed. Input of text is done phonically (using the spoken order consonant-subscript consonant-second subscript consonant-vowel-diacritic). The filter converts phonic description of consonantal clusters into a graphic TEXnical description of these. Thanks to TEX booleans, independent vowels can be automatically decomposed according to recent reforms of Khmer spelling. The last section presents a forthcoming implementation of Khmer into a 16-bit TEX output font, solving the kerning problem of consonantal clusters.

Introduction to Khmer Script

The Khmer script is used to write Khmer, the official language of the Cambodian Republic, and belongs to the Mon-Khmer group of Austroasiatic languages. It is a very old and beautiful script, and from the typesetter's point of view, one of the most challenging and exciting scripts in the world.

To understand the complications of Khmer typesetting, we will start with a quick overview of the Khmer writing system. Khmer is written from left to right; the Khmer alphabet has 32 *consonants*, the following:

ក ខ គ ឃ ង ច ឆ ជ ឈ ញ ដ ឋ ឌ ឍ ណ ត ថ ទ ធ
ន ប ផ ព ភ ម យ រ ល វ ស ហ ឡ

The character អ denotes the absence of a consonant. From the typesetter's point of view and with respect to collating order, it might as well be considered as a consonant. We will use a box □ to denote an arbitrary consonant.

These 33 "consonants" (except ឡ) can appear in the form of *subscript consonants:*

◌្ក ◌្ខ ◌្គ ◌្ឃ ◌្ង ◌្ច ◌្ឆ ◌្ជ ◌្ឈ ◌្ញ ◌្ដ ◌្ឋ ◌្ឌ
◌្ឍ ◌្ណ ◌្ត ◌្ថ ◌្ទ ◌្ធ ◌្ន ◌្ប ◌្ផ ◌្ព ◌្ភ ◌្ម ◌្យ
◌្រ ◌្ល ◌្វ ◌្ស ◌្ហ ◌្អ

A subscript consonant is pronounced after the "primary" consonant. Nevertheless, as the reader has certainly noticed, the subscript consonant ◌្រ is written on the left of the primary consonant.

It is also possible to have *two* subscript consonants carried by the same primary consonant. In that case, the second subscript consonant has to be ◌្រ. Examples: ង្ក្រ, ស្ត្រ.

A consonant, consonant + subscript or consonant + double subscript combination can carry a *vowel*. There are 28 vowels:

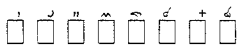
◌ា ◌ិ ◌ិះ ◌ី ◌ឹ ◌ឹះ ◌ឺ ◌ុ ◌ុះ ◌ូ ◌ួ
◌ើ ◌ើះ ◌ឿ ◌ៀ ◌េ ◌េះ ◌ែ ◌ែះ ◌ៃ ◌ោ
◌ោះ ◌ៅ ◌ុំ ◌ំ ◌ាំ ◌ះ ◌ៈ

Although vowels are always pronounced *after* consonants, their graphical representation literally surrounds the consonant/subscript combination: they can appear above, beneath, on the right or on the left of consonants. Often a vowel's glyph has two or three non-connected parts.

When combining vowels with subscript consonants, the following graphical rules are followed:

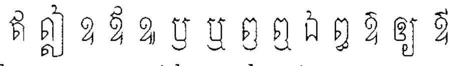
- if the subscript has a right protruding stem then the vowel ◌ា connects to the subscript and not to the consonant: ◌្ឈ + ◌ា = ◌្ឈា etc.
- if the consonant carries both a subscript ◌្រ and a vowel with left branch, then the latter is placed on the left of the former: ◌្រ + ◌េ = ◌្រេ etc.
- if the consonant carries both a subscript consonant and a subscript vowel, then the latter is placed underneath the former: ◌្ត + ◌ុ = ◌្តុ, ◌្រ + ◌ុ = ◌្រុ etc.

Finally, a group of characters as described above can carry a *diacritical mark*. These are always placed above the character:



We will call the combination of consonant and eventual subscript consonant, second subscript consonant, vowel and diacritical mark, a *consonantal cluster*. Theoretically there can be 535,060 different consonantal clusters, but in practice less than 1% of them are really used. An analytic decomposition of A. Daniel's Khmer-French dictionary (Daniel, 1985) has provided no more than 2,821 different consonantal clusters out of 25,000 entries; colloquial Khmer may require even less clusters.

Besides consonantal clusters there are also 14 "stand-alone" characters in the Khmer alphabet:



These carry neither subscript consonants, nor vowels, nor accents. They cannot be found in subscript form. Orthographical reforms of Khmer have in some cases replaced them by "regular" consonantal clusters.

Inside a sentence, Khmer words are *not* separated by blank space. A blank space denotes the end of a sentence (or of part of a sentence: it plays the role of the period or of the semicolon in Latin script).

Hyphenation occurs between *syllables:* a syllable consists of one or two consonantal clusters with the sole restriction that the second cannot have a vowel. When a word is hyphenated, a hyphen is used. Sentences are "hyphenated" into words, but in that case, no hyphen is used. So from the typesetter's point of view, between two clusters hyphenation can be

1. forbidden (when the two clusters belong to the same syllable);
2. allowed and producing a hyphen (when the two clusters belong to the same word);
3. allowed without producing a hyphen (when the two clusters belong to different words in the same sentence).

This quick overview of the Khmer script has shown some of its particularities (see also Daniel (1985 and 1992), Tonkin (1991) and Nakanishi (1980)). To conclude, the author would like to underline the fact that the main difficulty in Khmer typesetting is the divergence between phonic and graphical representation of consonantal clusters (see Figure 1).

This paper is divided into five sections:

1. the definition and discussion of an 8-bit encoding table for information interchange and storage in the Khmer script. Consonantal clusters are encoded according to their phonic representation;
2. the presentation of three Khmer font families, designed in the METAFONT language. These fonts correspond to the three main styles of Khmer type and provide sufficient metaness to perform optical scaling, continuous interpolation from light to extra-bold weight and strong raster optimization;
3. the description of the process of deriving the graphical representation of consonantal clusters out of the phonic one (this process being implemented in an ANSI C preprocessor);
4. an overview of hyphenation and spelling reform rules and their realization in the preprocessor;
5. shortcomings of the Khmer typesetting system and plans for future developments.

The author would like to thank Prof. Alain Daniel (Institute of Oriental Languages and Civilizations, Paris) for his continuous support and encouragement and the Imprimerie Louis-Jean (Gap) in the person of Maurice Laugier, for having financed this project.

An 8-bit Encoding Table for the Khmer Script

Discussion. As mentioned in the introduction, Khmer language is written using consonantal clusters and stand-alone special characters. The collating order of consonantal clusters is given lexicographically out of the cluster components:

Let $C_1 = c_1 s_1 s'_1 v_1 d_1$ and $C_2 = c_2 s_2 s'_2 v_2 d_2$ be two consonantal clusters, where $c_1, c_2 \in$ {consonants}, $s_1, s_2 \in \varnothing \cup$ {subscript consonants}, $s'_1, s'_2 = \varnothing$ or ្រ, $v_1, v_2 \in \varnothing \cup$ {vowels} and $d_1, d_2 \in \varnothing \cup$ {diacritics}. Then

1. $c_1 \succ c_2 \Rightarrow C_1 \succ C_2$;
2. if $c_1 = c_2$ then $s_1 \succ s_2 \Rightarrow C_1 \succ C_2$ (where $\varnothing$ precedes any other element);
3. if $c_1 = c_2$ and $s_1 = s_2$ then $s'_1 \succ s'_2 \Rightarrow C_1 \succ C_2$;
4. if $c_1 = c_2, s_1 = s_2$ and $s'_1 = s'_2$ then $v_1 \succ v_2 \Rightarrow C_1 \succ C_2$;
5. if $c_1 = c_2, s_1 = s_2, s'_1 = s'_2$ and $v_1 = v_2$ then $d_1 \succ d_2 \Rightarrow C_1 \succ C_2$.

The table of 128 codes for Khmer characters presented on the following page respects the collating order. Besides consonantal clusters and special characters, the following signs have been included in the 8-bit encoding:

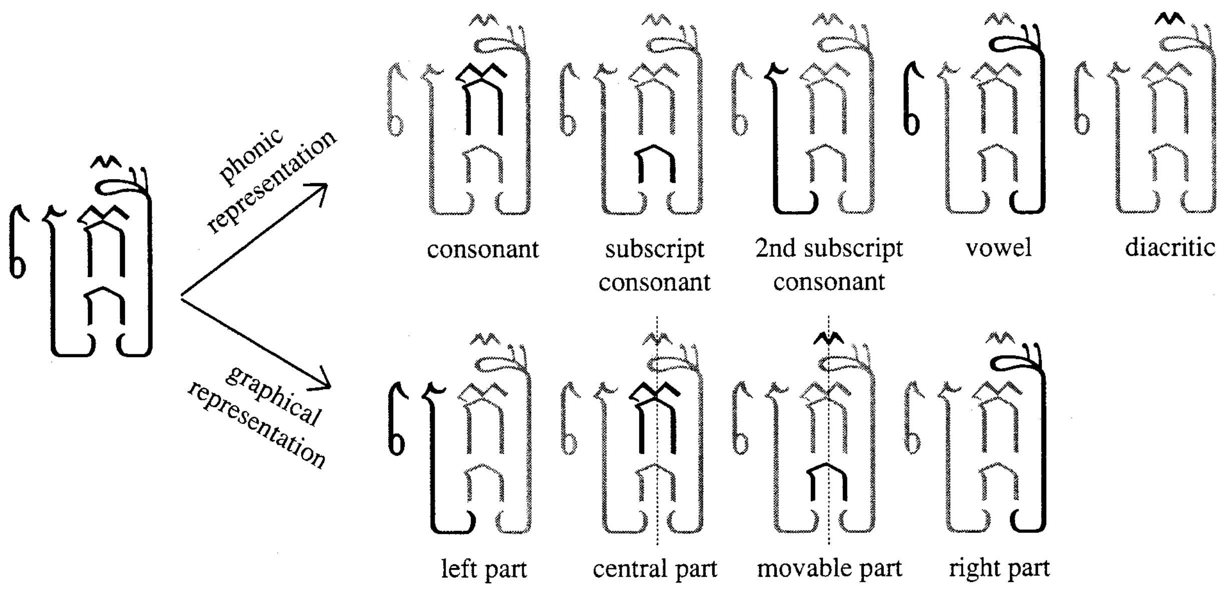


Figure 1: Decomposition of a Khmer consonantal cluster.

1. digits: ០, ១, ២, ៣, ៤, ៥, ៦, ៧, ៨, ៩;
2. punctuation marks other than the ones borrowed from Latin script: ៗ (leikto), a variant form of the digit ២, indicating that the previous word is repeated (similar to Latin *bis*), ។ (khan) and ៕ (bariyatosan), equivalent to a full stop, ៖ (camnocpikuh), a graphical variant of the Latin colon, and the French guillemets «, »;
3. the currency symbol ៛ (rial);
4. the invisible code WBK (word-break) to indicate the word limits inside a sentence.

   Have *not* been included in the table:

- the archaic characters ឣ and ឤ which were abolished about a century ago;
- the punctuation marks ៚ (cow's urine) and ៙ (coq's eye), used in poetry, devination and classical texts;
- the variant forms ◌ឹ, ◌ឹ of ◌ឹ, ◌ឺ, used in *Dictionnaire Cambodgien* (1962).

These characters are nevertheless included in the TEX output fonts and can be accessed via macros.

The table. The table of codes 128–255 of the proposed 8-bit encoding for Khmer information interchange and storage follows. The 7-bit part of the table conforms to ISO 646 (standard 7-bit ASCII). Positions 0xCF and 0xDF are empty.

"8*	"9*	"A*	"B*	"C*	"D*	"E*	"F*
ក	ថ	◌្ក	◌្ថ	អ	០	◌្អ	◌ៅ
ខ	ទ	◌្ខ	◌្ទ	ឥ	១	◌ា	◌ុំ
គ	ធ	◌្គ	◌្ធ	ឦ	២	◌ិ	◌ំ
ឃ	ន	◌្ឃ	◌្ន	ឧ	៣	◌ី	◌ាំ
ង	ប	◌្ង	◌្ប	ឩ	៤	◌ឹ	◌ះ
ច	ផ	◌្ច	◌្ផ	ឪ	៥	◌ឺ	◌ៈ
ឆ	ព	◌្ឆ	◌្ព	ឫ	៦	◌ុ	«
ជ	ភ	◌្ជ	◌្ភ	ឬ	៧	◌ូ	»
ឈ	ម	◌្ឈ	◌្ម	ឭ	៨	◌ួ	◌៉
ញ	យ	◌្ញ	◌្យ	ឮ	៩	◌ើ	◌់
ដ	រ	◌្ដ	◌្រ	ឯ	៛	◌ឿ	◌៌
ឋ	ល	◌្ឋ	◌្ល	ឰ	ៗ	◌ៀ	◌៍
ឌ	វ	◌្ឌ	◌្វ	ឱ	៖	◌េ	◌៎
ឍ	ស	◌្ឍ	◌្ស	ឳ	។	◌ែ	◌៏
ណ	ហ	◌្ណ	◌្ហ	ឲ	៕	◌ៃ	◌៊
ត	ឡ	◌្ត	WBK			◌ោ	◌័

Codes 0x80–0x9F and 0xC0 represent consonants; the corresponding subscript consonants are offset by 32 positions: they are represented by codes 0xA0 - 0xBE and 0xE0. The consonant 0x9F does not have a corresponding subscript consonant. The practice of subscripts having to be 32 positions apart from primary consonants is similar to the 32-position offset of uppercase and lowercase letters in ISO 646 (7-bit ASCII).

Codes 0xC0–0xCE represent special characters. Digits have been placed in positions 0xD0–0xD9, vowels in 0xE1–0xF5 and diacritics in 0xF8–0xFF. Finally, 0xFA is the currency symbol, 0xDB–0xDE are punctuation marks and 0xBF is the word-break code WBK.

Because of the 128-character limitation, vowels ◌ិះ, ◌ឹះ, ◌ុះ, ◌ើះ, ◌េះ, ◌ែះ, ◌ោះ have not been included in the table. They have to be represented by the following code pairs:

◌ិះ	=	0xE2 0xF4	◌ឹះ	=	0xE4 0xF4
◌ុះ	=	0xE6 0xF4	◌ើះ	=	0xE9 0xF4
◌េះ	=	0xEC 0xF4	◌ែះ	=	0xED 0xF4
◌ោះ	=	0xEF 0xF4			

Requirements for Khmer script software. As in the case of Arabic and Hindi, software displaying Khmer text has to provide context-analytic algorithms. Following is an exhaustive list of the necessary context-dependent transformations:

1. when code 0xBA follows a code in the range 0x80–0x9E, 0xC0 then their glyphs must be permuted, e.g., ក + ◌្រ → ក្រ.
2. when code 0xBA follows a pair of characters $\alpha\beta$, with $\alpha \in \{0x80\text{–}0x9E, 0xC0\}, \beta \in \{0xA0\text{–}0xBE, 0xE0\}$ then the glyph of 0xBA must appear on the left of the glyphs of α, β, e.g., ស + ◌្ត + ◌្រ → ស្ត្រ.
3. when codes 0xE9–0xEC and 0xEF–0xF0 follow a combination of character codes $\alpha, \alpha\beta, \alpha\beta\gamma$ where α and β are as in the previous item and γ =0xBA, then the glyph េ must appear on the left of the latter combinations. Example: ស + ◌្ត + ◌្រ + ◌ោ → ស្ត្រោ.
4. when codes 0xED and 0xEE follow a combination $\alpha, \alpha\beta, \alpha\beta\gamma$ of codes as in the previous item, then their glyphs must appear on the left of these combinations;
5. when code 0x89 (ញ) is followed by a code in the range 0xA0–0xBE, 0xE0 then the variant glyph ញ must be used. Example: ញ + ◌្ជ → ញ្ជ.

When the second code is 0xA9 then a variant glyph must be used for it as well: ញ + ◌្ញ → ញ្ញ.

These contextual transformations have been implemented by the author into a modified version of the Macintosh freeware text editor Tex-Edit by Tim Bender, included in the package. In Figure 2 the reader can see the effect of striking successively keys <ក>, <◌្ប> (<subscript modifier> followed by <ប>), <◌្រ> (<subscript modifier> followed by <រ>), <◌ោ>, to finally obtain the consonantal cluster ក្ប្រោ.

METAFONTing Khmer

Font styles. There are three styles used in Khmer typesetting: standing (aksar ch-hor), oblique (aksar chrieng) and round (ak-sar mul). The latter is virtually identical with inscriptions of the 12th and 13th centuries at Angkor Wat and is reserved for religious texts, chapter headings, newspaper headlines, inscriptions and on other occasions where it is wished to make a contrast with the oblique script, to add a touch of formality, or to provide variation of emphasis (see Tonkin (1991)).

The author has designed three METAFONT font families, corresponding to these styles; samples of these fonts in 14.4 point size can be seen on Figure 3; Figure 4 shows a sample headline using the round font.

In Figure 5, the Khmer letter ក has been reproduced 256 times, with different values of two parameters: the widths of "fat" and "thin" strokes. The central vertical symmetry axis represents "Égyptienne"-like characters, where the parameters have the same value. This classification can of course be refined and allows an arbitrarily precise choice of the font gray density.

Obtaining a Graphical Representation out of a Phonic One

Above we have given a quick overview of the (minimal) contextual analysis involved in displaying Khmer script on screen. The situation is much more complicated in the case of high quality typesetting.

TeX is the ideal tool for typesetting in Oriental scripts like Khmer, because of the inherent fundamental concept of *boxes* (see *The TeXbook* (Knuth, 1989) and Kopka (1991 and 1992)). As in mathematical formulas, elements of a consonantal cluster are moved to aesthetically correct positions and then grouped into a single and indivisible "box" which TeX treats as a single entity.

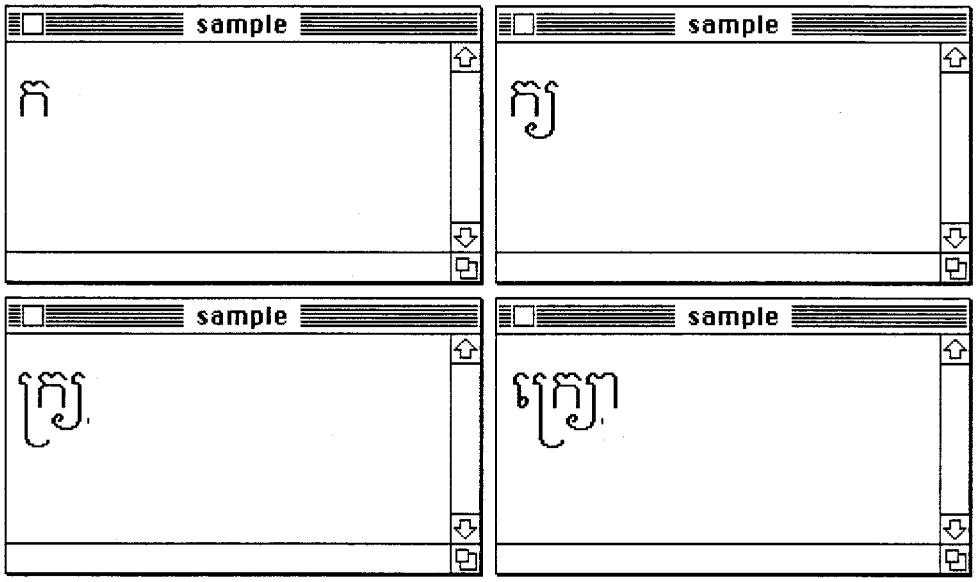


Figure 2: A text editor with Khmer contextual properties.

Standing characters

នៅឆ្នាំ ១៩៦២ ប្រទេសកម្ពុជាបានចេញមក នៅសល់តែស្បែកនឹងឆ្អឹង អំពីការប្រយុទ្ធដ៏យូរអង្វែង
នឹងដ៏ឃោរ ឃៅមួយ ប្រឆាំងនឹងការជិះជាន់របស់យួន នឹងសៀម ។ ការឈឺចាប់នៃប្រជាពលរដ្ឋយើង-
នៅពេលនេះ គឺហួសពីការគិតទៅហើយ ប៉ុន្តែពួកបស្ចិមប្រទេស គេនៅតែពុំដឹងឮ ចំពោះការឈឺចាប់-
នេះទេ ។ ចំណែកខ្មែរវិញ យើងមិន មានភ្លូបសោះឡើយ ។

Oblique characters

*នៅឆ្នាំ ១៩៦២ ប្រទេសកម្ពុជាបានចេញមក នៅសល់តែស្បែកនឹងឆ្អឹង អំពីការប្រយុទ្ធដ៏យូរអង្វែង នឹងដ៏-
ឃោរ ឃៅមួយ ប្រឆាំងនឹងការជិះជាន់របស់យួន នឹងសៀម ។ ការឈឺចាប់នៃប្រជាពលរដ្ឋយើងនៅ-
ពេលនេះ គឺហួសពីការគិតទៅហើយ ប៉ុន្តែពួកបស្ចិមប្រទេស គេនៅតែពុំដឹងឮ ចំពោះការឈឺចាប់នេះទេ
។ ចំណែកខ្មែរវិញ យើងមិន មានភ្លូបសោះឡើយ ។*

Round characters

**នៅឆ្នាំ ១៩៦២ ប្រទេសកម្ពុជាបានចេញមក នៅសល់តែស្បែកនឹងឆ្អឹង អំពីការប្រយុទ្ធដ៏យូរអ-
ង្វែង នឹងដ៏ឃោរ ឃៅមួយ ប្រឆាំងនឹងការជិះជាន់របស់យួន នឹងសៀម ។ ការឈឺចាប់នៃប្រ-
ជាពលរដ្ឋយើងនៅពេលនេះ គឺហួសពីការគិតទៅហើយ ប៉ុន្តែពួកបស្ចិមប្រទេស គេនៅតែពុំ-
ដឹងឮ ចំពោះការឈឺចាប់នេះទេ ។ ចំណែកខ្មែរវិញ យើងមិន មានភ្លូបសោះឡើយ ។**

Figure 3: Samples of Khmer fonts.

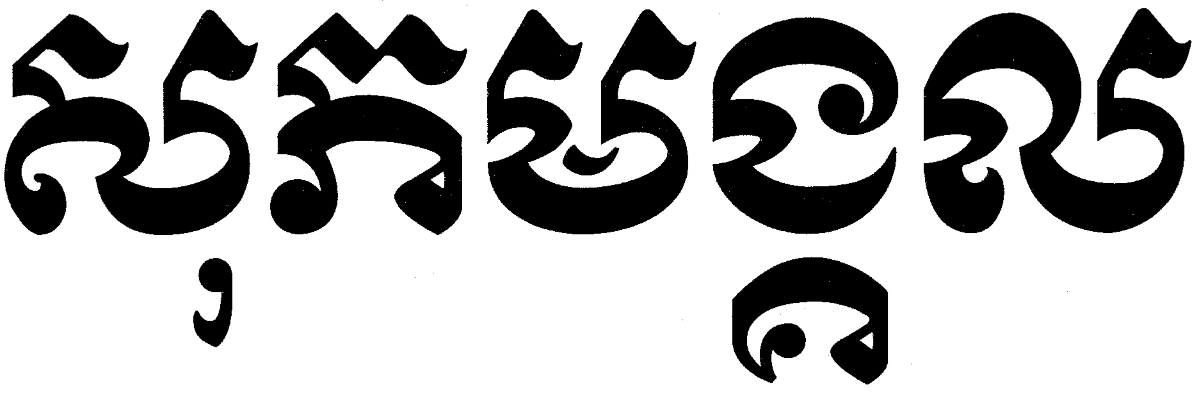


Figure 4: Headline in round style.

In this section we will see how the graphical representation of a cluster is constructed, using both the preprocessor and TEX.[1]

Graphical classification of Khmer cluster components. As already mentioned, there is a strong divergence between the phonic and graphical representation of a consonantal cluster: for example, if c =ឈ៊, s_1 =◌្ឈ, s_2 =◌្រ, v =េ◌ា, then for the same cluster ឈ៊្ឈ្រោ, the former representation is <c><s_1><s_2><v> and the latter <v (left branch)><s_2><c><s_1><v (right branch)>.

A thorough study of Khmer script and traditional typography has resulted in the following classification of graphical components of a consonantal cluster:

1. **the "left part".** Four elements which are placed on the left of a consonant: េ◌, ែ◌, ៃ◌, ◌្រ.
2. **the "central part".** All consonants: ក, ខ … អ. Also consonant + vowel ∈ {◌ា, ◌ាំ, ◌ៅ} combinations, whenever the vowel is attached to the consonant and not to a subscript: កា, កាំ, កៅ etc. but *not* ក្បា.

   The difference between "left" and "central" part is that only the latter is taken into account when determining the symmetry axis of the cluster.
3. **the "movable" part.** Subscripts and superscripts which are moved horizontally so that their symmetry axis coincides with the axis of the central part: ◌្ក … ◌្អ, and ◌ិ, ◌ី, ◌ឹ, ◌ឺ, ◌ុ, ◌ូ, ◌ំ, ◌៉, ◌៊, ◌៌, ◌៍, ◌៎.
4. **the "right" part.** Elements placed on the right of the central part, and not involved in the determination of the cluster symmetry axis. In this category we have certain subscript characters: ◌្ឃ … ◌្ញ, as well as selected subscript and superscript vowels and diacritical marks: ◌ួ, ◌ឿ, ◌ៀ, ◌ះ, ◌ៈ, ◌់, ◌៍, ◌៏.

The effective graphical construction of a consonantal cluster by TEX is done in the following way: the preprocessor's output replaces the phonic representation of a cluster (in the encoding described under Discussion) by a TEX macro `\ccl` with 5 arguments: the first is a 9-digit number representing the phonic representation of the cluster (and with the property that if N, N' are numbers representing clusters C, C' then $C \succ C' \Leftrightarrow N > N'$, where $\succ$ is the collating order of clusters and $>$ the usual ordering of integers); the remaining four correspond to the four parts of the graphical decomposition of a cluster as described above. For example,

```
\ccl{050311501}{e/r}{gA}{/k}{'}
```

indicates a left part `e/r` (េ◌្រ), a central part `gA` (ឃ៊ា), a movable part `/K` (◌្ក) and a right part `'` (◌់). This example illustrates the important fact that the symmetry axis of the central part is not necessarily the middle axis of the box containing the central part:

ឃ៊្កា and not ឃ៊្កា

The difference is of more than just aesthetic nature: in some cases the vertical alignment of elements of a cluster is necessary to determine the cluster itself. Take for example characters 0x89 (ញ) and 0x96 (ញ). When the latter is followed by a vowel ◌ា it becomes ញា, which is indistinguishable from the upper part of the former: it is the lower part

[1] Theoretically it would be possible for TEX to graphically represent a cluster out of its phonic representation without the assistance of a preprocessor. However, this process would be too slow and memory-consuming for real-life typesetting.

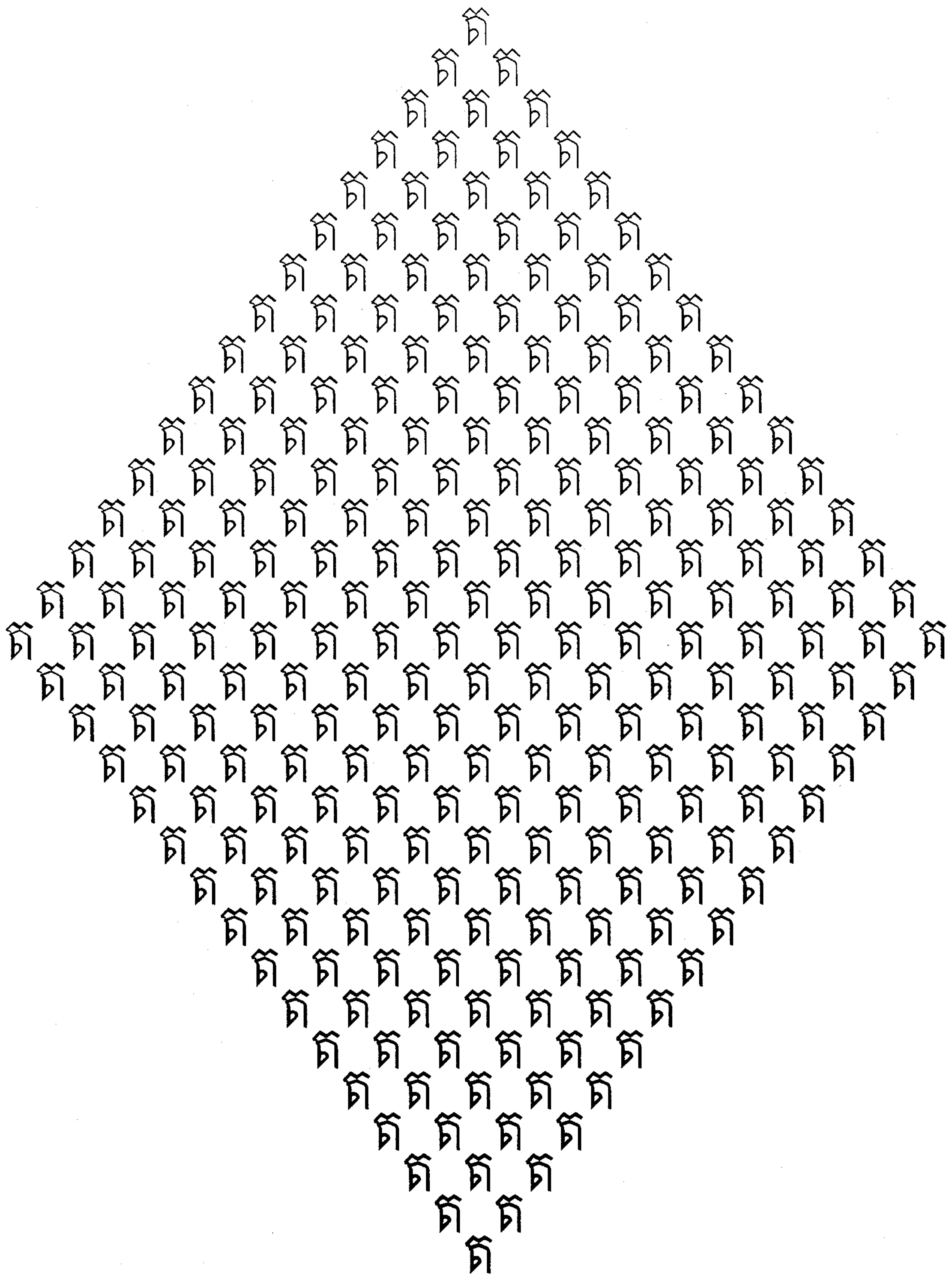


Figure 5: Test table for gray density fine-tuning of Khmer font.

◌ុ that allows differentiation. But when both happen to carry the same subscript consonant then this lower part vanishes. The difference will be found in the alignment of the subscript consonant: in the case of ញ one would have for example ញ្ក, while in the case of ញ it would be ញ្ក.

From these considerations we conclude that the symmetry axis location is a vital piece of information for every character; it depends on the shape of the individual character and *cannot* be given by a general font-independent rule.

In TEX one has several global parameters for a given font, but only 4 for every individual character of the font: width, height, depth, italic correction. The author has used the parameter "italic correction" as a carrier of the information on the symmetry axis location.

The construction mechanism is very simple: TEX typesets first the left part and the central part of the cluster; then it moves to the left, by an amount equal to the italic correction of the central part and typesets the movable part; finally it moves back to the right edge of the central part and typesets the right part of the cluster.

To simplify this mechanism, all movable elements are of zero width. The reader can see an example in Figure 6, where TEX boxes are displayed in gray and the symmetry axis of the central part by means of a dotted line.

Special cases and exceptions. The mechanism of cluster construction described above fails in certain special cases. These are handled by using variant forms of graphical elements. Following is a quick description of these cases.

1. often two or three subscripts or superscripts are found in the same cluster. In these cases the following rules apply:
   (a) in the case of two subscript consonants, the second being necessarily ◌្រ, a deeper form of the latter is used: ◌្រ + ◌្ក = ◌្ក្រ;
   (b) in the case of a subscript consonant and a subscript vowel, the vowel is placed under the subscript consonant: ◌្ក + ◌ួ = ◌្កួ. This rule applies also for the subscript consonant ្រ: ◌្រ + ◌ួ = ◌្រួ;
   (c) in the case of two subscript consonants and a subscript vowel, the consonants are placed as in (a) and the vowel is placed on the right of ្រ: ◌្រ + ◌្ក + ◌ួ = ◌្ក្រួ;
   (d) in some cases we have both a superscript vowel and a diacritical mark. The following combinations are known: ◌៊ី, ◌៊ឹ, ◌៊ី, ◌៊ឺ, ◌៊ំ, ◌៊៍;
2. to prevent confusion between the letter ប followed by vowel ◌ា, and the letter ហ, the former combination of consonant and vowel is written បា. A variant of this letter is used in the presence of a subscript: បា + ◌្ក = ប្កា;
3. when a cluster with ◌្រ contains vowel ◌ឿ or ◌ៀ, then the width of the primary consonant determines the depth of the vowel: ទ + ◌ៀ + ◌្រ = ទ្រៀ, but ឈ + ◌ៀ + ◌្រ = ឈ្រៀ;
4. the letter ឡ is not supposed to carry a subscript consonant; in some rare cases, it carries subscript ◌្ស: ឡ្ស.

Collating order. As mentionned in the previous section, the TEX command `\ccl`, obtained by the preprocessor, describes a cluster by means of five arguments. The last four arguments describe the cluster graphically: they correspond to the four parts of the graphical decomposition of a cluster, according to the subsection on Graphical classification of Khmer cluster components. The first argument corresponds to the phonic decomposition of the cluster; it is a 9-digit number $N = c_1c_2s_1s_2s_3v_1v_2d_1d_2$ where

1. c_1c_2 determines the primary consonant of the cluster: c_1c_2 goes from 01 = ក, to 33 = អ;
2. s_1s_2 determines the (first) subscript consonant: s_1s_2 = 00 if there is no subscript consonant, otherwise s_1s_2 goes from 01 = ◌្ក, to 32 = ◌្អ;
3. $s_3 = 0$ if there is no second subscript consonant, 1 if there is a second subscript ◌្រ;
4. v_1v_2 determines the vowel: v_1v_2 = 00 if there is no vowel, otherwise v_1v_2 goes from 01 = ◌ា, to 28 = ◌ះ;
5. d_1d_2 determines the diacritical mark: d_1d_2 = 00 if there is no diacritic, otherwise d_1d_2 goes from 01 = ◌៉, to 08 = ◌៍.

A complete list of characters, alphabetically ordered, is given in the Introduction. From the rules of collating order, it follows that for clusters C, C' and their corresponding 9-digit numbers N, N', we have

$$C \succ C' \Leftrightarrow N > N'$$

where $\succ$ is the collating order of clusters. The numbers N, N' can be easily ordered since the collating order of clusters corresponds to their order as integers. This fact allows straightforward searching, sorting, indexing and other collating order involving operations.

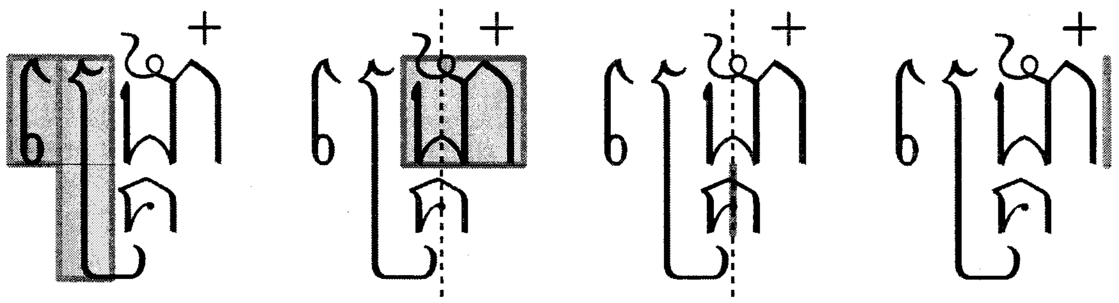


Figure 6: Construction of a Khmer consonantal cluster by TEX.

Hyphenation and Other Preprocessor Features

Hyphenation. Hyphenation of Khmer obeys a very simple rule: words are hyphenated between syllables. Unfortunately this rule can hardly be implemented by a computer since there is no algorithmic way of detecting syllables: a syllable can consist of one *or two* consonantal clusters.

With the help of Prof. Alain Daniel an empirical hyphenation mechanism has been developed, out of several general rules and observations. Following is a first set of rules — there will be further refinement after thorough testing on bigger amounts of Khmer text.

Let C, C' be consonantal clusters. Hyphenation C-C' is possible whenever:

1. C' contains a vowel;
2. C contains a vowel among ◌ិះ, ◌ឹះ, ◌ុះ, េ◌ឹះ, េ◌ះ, ែ◌ះ, ៃ◌, ោ◌ះ, េ◌ី, ◌ុំ, ◌ំ, ◌ះ, or one of the diacritical marks ◌់, ◌៍;

Hyphenation is always possible before or after special characters.

TEX provides an internal hyphenation mechanism based on *hyphenation patterns*. Unfortunately this mechanism cannot be used in the case of Khmer consonantal clusters, since these are enclosed in boxes and hence cannot be considered as characters by TEX. For this reason, the hyphenation algorithm is performed by the preprocessor; whenever one of the two rules above is satisfied, the TEX macro \- is included in the output. This command expands as

```
\def\-{\discretionary{-}{}{}}
```

so that a hyphen is obtained whenever a word is hyphenated. There is no algorithm yet for automatic decomposition of sentences into words: the user is asked to include WBK (word-break) codes between words inside a sentence. These codes are converted into \wbk commands by the preprocessor; \wbk expands into

```
\def\wbk{\discretionary{}{}{}}
```

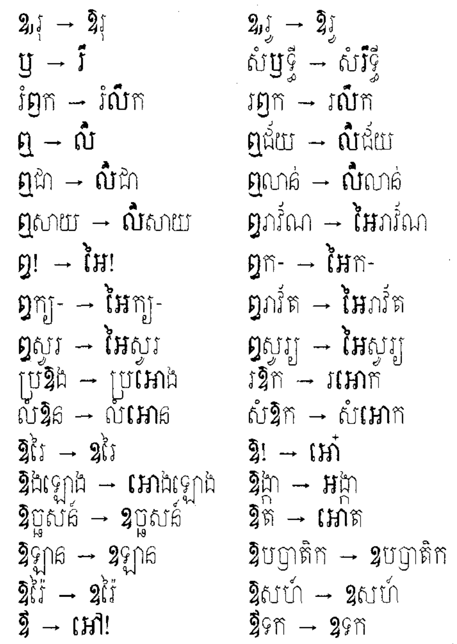
that is: a potential hyphenation point, *without* hyphen.

Decomposition of special characters and spelling reforms. The special characters (codes 0xC1–0xCE) are mostly historical residues and loans from other languages (Pali and Sanskrit). There have been many attempts by the Cambodian Ministry of Education to restrain their number, by eventually replacing some of them with regular consonantal clusters.

This replacement can vary from word to word. Prof. Alain Daniel has established a list of reformed words and their replacements. This list is known by the preprocessor, which will output every special character as a TEX macro with a numeric argument, indicating the potential replacement by some other special character or by a consonantal cluster. For example, according to the surrounding word, ឱ is output as \ao0, \ao1, \ao2, \ao3 or \ao4. If a certain boolean variable \ifreformed is false then all five macros will always expand into ឱ. On the other hand, if the boolean is true, then the first macro will expand into ឱ, the second into អោ, the third into ឲ, the fourth into អ and the fifth into អៅ.

Following is a first list of reformed words, known by the preprocessor. The special characters and their decompositions are set in bolder type.

រំឭល → រំ**អឺ**ល	រ**ឭ**ល → រ**អឺ**ល
ឭសុការ → **ឱ**សុការ	**ឮ**ស → **ឭ**ស
ឮសធរ → **ឭ**សធរ	**ឮ**សាន → **ឭ**សាន
ឮស្វរ → **ឭ**ស្វរ	**ឮ**ស្វរៈ → **ឭ**ស្វរៈ
ឱក → **អុ**ក	**ឱ**ស → **អុ**ស
ក្រ**ឱ** → ក្រ**អៅ**	ក្រាំងក្រ**ឱ** → ក្រាំងក្រ**អៅ**
ឲកា → **ឱ**កា	**ឲ**ដ្ឋ → **អូ**ដ្ឋ
ឲន → **អូ**ន	**ឲ**ម! → **ឱ**ម



Shortcomings and Plans for Further Development

The system presented in this paper allows high quality Khmer typesetting. It is the first Khmer typesetting system resolving problems such as text input in phonic order, positioning of subscripts and superscripts, optical scaling, hyphenation and replacement of special characters.

Nevertheless the graphical cluster-construction algorithm as described in this paper has certain flaws; a few examples:

- if a consonant with subscript consonant carries the ុ vowel, then the latter should be justified at the right edge of the *subscript*, which is not necessarily aligned with the right edge of the consonant. For example, in the (hypothetical) cluster រ្ណុ៌, the ុ is badly positioned;
- take a narrow letter (like រ, រ៌) which carries a large subscript (like ្ណ or ្ឈ) and suppose you are at the line boundary (either left or right); then contrary to the normal use of subscripts, it is the subscript which should serve for line justification, and not the consonant.

These problems cannot be solved using the current mechanism (in which TEX considers that all subscripts and superscripts are of zero width). It could be possible to use subscripts with non-zero width, but (a) this would slow the process down, and (b) it wouldn't solve the problem of the line boundary, since we are asking for contradicting properties: inside a sentence subscripts should not interfere in determining the distance between clusters, while at the line's boundary they should.[2] Furthermore, one could imagine a sentence ending with រ្ណ៌ and the next sentence starting with រ្ណ. The blank space in-between is hardly sufficient to prevent clusters from overlapping... visually the beginning of the sentence is lost.

Corrections to these problems can be done manually (after all these problems occur very rarely). But a much more natural and global solution would be to treat consonantal clusters as individual codes in a 16-bit encoding scheme. As mentioned in the introduction, only 2,821 clusters (out of 535,000 theoretical possibilities) have been detected in the fairly complete dictionary of Prof. Alain Daniel, so a 16-bit table would be more than sufficient to cover them.

Text input could still be done using the 8-bit encoding of Section 1; the preprocessor would then convert the 8-bit description of consonantal clusters into their codes in the 16-bit table (or by their explicit construction, if for any reason they are not included in the table). This approach is similar to Kanji construction out of Kana characters in Japanese, or to Hangoul construction out of elementary strokes in Korean.

An extended version of TEX —which according to D.E. Knuth's directives should not be called TEX anymore— would then perform real kerning and hyphenation, since in this case consonantal clusters would be treated by TEX as *letters*. Work in the direction of an extended TEX is currently being done by Philip Taylor[3] and his team (NTS project), and by John Plaice[4] (Ω project)[5]

[2] Unfortunately, in TEX there is no such thing as an `\everyline` command.

[3] Royal Holloway College (UK)

[4] Université Laval (Canada)

[5] It should be mentioned that the Japanese TEX Users Group, and the ASCII Corporation, Kanawasakishi Kanagawa, Japan, have developed and released 16-bit versions of TEX. Unfortunately, these are not "real" 16-bit TEXs (and hence inefficient for 16-bit Khmer), because they allow only 256 different character widths (it happens that Japanese Kanji, just like Chinese characters, have all the same width) and 256^2 kerning pairs or ligatures. True 16-bit TEX should allow $256^2 = 65,536$ character widths, and $65,536^2 = 4,294,967,296$ kerning pairs or ligatures. Also 16-bit hyphenation patterns should be possible. Besides Khmer, such an extended TEX version would be extremely useful for ligatured Arabic Naskhi, Urdu Nastaliq, Hebrew with cantillation marks, and other scripts with "advanced typographical requirements".

By using virtual property lists, no additional bitmap files would be added. The 16-bit font would be made by virtually assembling glyphs taken from the already available 8-bit font.

Availability

The METAFONT, TEX and C sources of all software presented in this paper belong to the **public domain.** They constitute a proposal for a Khmer *TEX Language Package*, submitted to the Technical Working Group on Multiple Language Coordination of the TEX Users Group and will be released after ratification. The α version of the package is currently being tested in Cambodia, and can be obtained by the author (`yannis@gat.citilille.fr`). This work will be presented either by Alain Daniel or by the author at the *First Conference on Standardization of Khmer Information Interchange*, organized by UNESCO, July 20–23 in Phnom Penh, Cambodia.

Bibliography

Daniel, Alain. *Dictionnaire pratique cambodgien-français.* Institut de l'Asie du Sud-Est, Paris, 1985.

Daniel, Alain. *Lire et écrire le cambodgien.* Institut de l'Asie du Sud-Est, Paris, 1992.

វចនានុក្រមខ្មែរ, ការផ្សាយរបស់ពុទ្ធសាសនបណ្ឌិត្យ, ព. ស. ២៥០៥ [*Dictionnaire Cambodgien.* Éditions de l'Institut Bouddhique, Phnom-Penh, 1962.]

Knuth, Donald E. *The TEXbook.* Computers and Typesetting, Volume A. Addison-Wesley, Reading, 1989.

Knuth, Donald E. *The* METAFONT*book.* Computers and Typesetting, Volume C. Addison-Wesley, Reading, 1986.

Kopka, Helmut. *LATEX, eine Einführung.* 3rd edition, Addison-Wesley, München, 1992.

Kopka, Helmut. *LATEX, Erweiterungsmöglichkeiten.* 3rd edition, Addison-Wesley, München, 1991.

Nakanishi, Akira. *Writing Systems of the World.* Charles E. Tuttle Company, Tokyo, 1980.

Tonkin, Derek. *The Cambodian Alphabet.* Transvin Publications, Bangkok, 1991.

Language-Dependent Ligatures

John Plaice
Département d'informatique, Université Laval, Ste-Foy, Québec, Canada G1K 7P4
plaice@ift.ulaval.ca

Abstract

The concept of *character cluster* is introduced to TEX. Any sequence of letters can be clustered to form a single entity, which can subsequently be used as a single character. The standard example for a cluster is a letter with diacritic marks. Clusters are introduced by extending the TEX input language and by modifying the TEX program. They can be used to define ligatures within TEX, without passing through the `.tfm` files, thereby allowing different languages to be typeset with different ligatures. Clusters can be used in hyphenation tables, thereby eliminating the need to have precomposed characters in fonts to have correct hyphenation. A single 256-character font becomes suitable for typesetting all the world's languages that use the Latin alphabet.

The Latin alphabet is the most widely used alphabet in the world. Although at first glance, it has only twenty-six letters, a more careful look will show that these letters can be used with myriads of different diacritic marks, and that some languages actually use special characters above and beyond the original 26. If we consider all the possible precomposed characters, Haralambous (*TUGboat*, 1989) counts 190 different symbols above the original 26, not including the double-diacritic Vietnamese characters (close to 100 all by themselves). In addition, Jörg Knappen's font for African languages uses another 100 or so characters (Haralambous, 1992a). Finally, the ISO-10646 Universal Character Encoding Standard includes close to 900 precomposed Latin characters.

If all Latin precomposed characters are encoded separately, then several families of 256-character fonts would be needed. This would be an incredible waste of space, as most characters would be re-encoded over and over again. Furthermore, most sites would end up stocking only one family of fonts, for the "important" languages (read English and West-European). On top of these considerations, the result would still not be technically desirable. It is standard typographic practice to place diacritical marks differently in different countries; in fact the marks may well look different from one country to another. Let's precompose all *those* combinations!

Another possibility is to use virtual fonts for every language. But, according to the Summer Institute for Linguistics's *Ethnologue*, there are over 6000 languages on this planet, and most of them use the Latin alphabet. To encode all the virtual fonts would literally require thousands of files, and few sites (if any) could afford the Brobdingnagian quantities of disk space required.

The only reasonable solution is to encode, in the fonts, diacritical marks separately from the letters that they are placed on. However, TEX currently only offers two ways to combine characters: ligatures and the `\accent` primitive. Ligatures do not (currently) do vertical placement and `\accent` has a fixed algorithm for character positioning. There is another alternative that can be considered: active characters. However, these can interact in strange ways with macros. Finally, preprocessors have a tendency to do strange stuff with macros.

In fact, what is needed is some sort of "active character" mechanism for *after* macro expansion, i.e., some sort of generalized ligature system that allows vertical, in addition to horizontal, displacement, and that does not require these ligatures to be encoded in the `.tfm` files.

In addition to not allowing vertical displacement, ligatures have their own problems. Jackowski and Ryćko (1992) discuss in detail the problems in using ligatures to access Polish characters with ogoneks. First, because implicit kerns and ligatures are defined one character at a time, it is very difficult, if at all possible, to correctly compose the characters and kern between these newly composed characters and their neighbors. Second, ligatures make it difficult for majuscule letters with diacritical marks to be given the appropriate space factor (`\sfcode`) codes. Third, if ligatures are used in hyphenation patterns, then the `\lefthyphenmin` and `\righthyphenmin` parameters do not work properly, as the individual characters in the ligatures are counted individually, rather than collectively. The basic problem is the same in all these cases: the generated ligatures are not treated as a single character but, rather, as a sequence of individual characters.

It is not just a system of generalized ligatures that is required, but also the ability to treat these new entities as single characters. This is done in an extension of TEX called Ω, using *character clusters*, which do exactly what is required above. A character cluster is a sequence of letters that can be considered to be a single character. A character cluster can be given an `\sfcode`, `\lccode` and `\uccode`, and can be used in the definition of hyphenation patterns. The typesetting of a character cluster is defined by an arbitrarily complex sequence of TEX instructions. With this added functionality, it becomes possible, to the best of our knowledge, to have a single 256-character font that is sufficient for encoding all the world's languages that use the Latin alphabet.

Character Clusters

A (character) cluster is an ordinary sequence of TEX instructions, which can be considered, for kerning and hyphenation purposes, as a single character, along with a name. The name is just a sequence of characters.

Clusters are defined using the `\defcluster` command. For example,

```
\defcluster{e'}{\'e}
```

defines a possible sequence of instructions to type é (using the dc fonts). Similarly,

```
\defcluster{ij}{\char"BC}
```

gives the Dutch **ij** ligature in the dc fonts.

Clusters are used using the `\cluster` command. For example,

```
\cluster{e'}t\cluster{e'}
```

would give the French word for summer, 'été'. Of course, no one would want to type `\cluster` all the time. To reduce typing problems, we introduce a new syntactic form: $<c_1c_2\ldots c_n>$ is equivalent to `\cluster{`$c_1c_2\ldots c_n$`}`. The above word therefore becomes `<e'>t<e'>`.

The traditional ligatures of TEX are really just alternative presentations of the composing characters. These can be presented as follows: $<c_1c_2\ldots c_n \mid c_{n+1}c_{n+2}\ldots c_m>$ means that the cluster $<c_1c_2\ldots c_n>$ can be broken up by the hyphenation algorithm into the basic characters $<c_{n+1}c_{n+2}\ldots c_m>$. For example, the '**ffl**' ligature can be represented by `<ffl|ffl>`.

Context Dependent Analysis

Now even the word `<e'>t<e'>` is too much work to type. One should be able to type an ordinary stream of text and, with no special instructions inserted, have the appropriate clusters inserted into the text. Of course, what is appropriate will depend significantly on the language and the family of fonts being used.

In the case of our example, it should be possible to simply type `e'te'` and have the system handle the rest. To do this, the `Chief Executive` routine is modified so that it filters through a deterministic finite state automaton (DFA) all the text that is in horizontal mode. This DFA changes depending on the language being typeset, the typesetting rules in effect and the font families being used.

For example, in French, many words use the œ and æ ligatures that were common for the typesetting of Medieval Latin (Becarri, 1992). A careful perusal of the medium-size *Petit Robert* dictionary (Rey & Rey-Debove, 1984) allowed the definition of a Lex-like set of patterns that can be used to determine when ae and oe should form the ligatures and when they should not:

```
^oe             <oe>
oe/ll           oe
oe/[cilmrstu]   <oe>
^aethuse        aethuse
mae             mae
ae              <ae>
```

This set of patterns only considers common terms. For names and persons, the set is wrong: for example, many Flemish and Dutch proper names have unligatured ae's and oe's that would be ligatured by this set of patterns.

The line

```
oe/[clmrstu] <oe>
```

should be read as if the letters oe are followed by any of `clmrstu`, and then those two letters are replaced by the <oe> cluster. Therefore the word `oeil` becomes `<oe>il`. The ^ refers to the beginning of a word.

This same mechanism can be used to handle words with strange hyphenation. For example, the German words `backen`, `Bettuch`, `Schiffahrt` could be written as follows

```
^backen      ba\disc{k-}{k}{ck}en
^Bettuch     Be\disc{tt-}{t}{t}uch
^Schiffahrt  Schi\disc{<ff>-}{f}{ff}ahrt
```

where the `\disc` means a discretionary break.
Similarly, the English word `eighteen` could become

```
^eighteen   eigh\disc{t-}{t}{t}een
```

Kerning

Kerning is a purely visual phenomenon, and so is handled after the context-dependent analysis has been made. In this situation, kerning becomes much simpler than the way that it is currently used in TEX and METAFONT. All that is required is to specify the kerning that must take place between any pair of characters or clusters. A Lex-like syntax is also used to specify these pairs.

The example given in the appendix gives the kerning in the `roman.mf` file for the Computer Modern family of fonts (Knuth, 1987), modified so that the letters **a**, **e** and **o** can each receive acute, grave, trema (or umlaut), and circumflex accents.

Hyphenation

Hyphenation patterns are no different from the old ones, except that they also allow clusters. However, it now becomes possible to have a period (`.`) in the middle of a pattern, as is required by some African languages, since the beginning of a word is marked by ^ and the end of a word by $. Accented characters are of course represented by clusters. For the purposes of `\lefthyphenmin` and `\righthyphenmin`, clusters are counted as a single character. The following example is an excerpt from the 8-bit hyphenation file `ghyphen3.tex` for German (Schwarz, 1990):

```
.kraf6
.k^^fc5ra
.lab6br
.liie6
.lo6s5k
.l^^f64s3t
```

which is replaced with:

```
^kraf6
^k<u">c5ra
^lab6br
^liie6
^lo6s5k
^l<o">4s3t
```

The second form is more readable and has the advantage of not being tied to a particular font encoding. It is now possible to separate the input encoding from the output encoding.

Handling 16-bit and 32-bit Input

Currently, there are many discussions about how to best handle 8-bit input. There are three major currently-used 8-bit extensions of ISO-646 (ASCII): ISO-8859-1 (Latin-1), Macintosh and IBM-PC. The users of all these systems would like to write using the characters that they have available. How are these needs to be reconciled with Ω? Furthermore, how should ISO-10646 and UNICODE 1.1 be handled? What is the relationship between these character encodings and the clusters?

The answer is quite simple. Internally, if a cluster has a corresponding encoding in ISO-10646, then that number is used for that cluster. However, if a cluster does not have a corresponding encoding, then a negative number is used (remember, ISO-10646 is really only a 31-bit encoding, and 'hi-bit on' is reserved for user-defined characters). The result is to separate input, internal and output encodings.

Implementation

At the time of writing, none of what has been proposed has been implemented, so many of the proposals are not finalized. Nevertheless, some detailed analysis has been completed. There are two parts to the implementation. First, the files defining the context-dependent analysis and the kerning must be translated into finite state automata readable by the `INITEX` program. Second, the TEX program must be changed to include a new data structure for clusters, new syntactic entities, as well as the basic routines which must be written or changed. Most of the work takes place in the `Chief Executive`.

Future Work

Character clusters are not just designed for the Latin alphabet. The same principles could be used to design compact Greek (Mylonas & Whitney, 1992) and Cyrillic fonts. More generally, typesetting Arabic (Haralambous, 1992a–c) or South Asian (Mukkavilli, 1991) scripts requires significant amounts of context analysis to choose the right variant of character and the correct set of ligatures to form a word and, in the case of Arabic, to correctly place vowels. Different solutions have been proposed, either using active characters, ligatures in the fonts or preprocessors, but none of them is sufficiently general. Character clusters should yield an elegant solution for these scripts.

Problems which have not been discussed here include the direction of text. At least four systems are used currently in different languages: left-right - top-down, right-left - top-down (Arabic, Hebrew, Syriac (Haralambous, 1991)), top-down - right-left (Japanese (Hamano, 1990)), and top-down - left-right (Mongolian (Haralambous, 1992a)). Some languages, such as Japanese, Mongolian and Epigraphical Greek, can be written in more than one direction. Clusters would be useful in implementing these questions.

Acknowledgements

The Ω project was devised by Yannis Haralambous and myself. It attempts to resolve some of the fundamental issues that have been raised during the discussions in the Technical Working Group on Multiple Language Coordination.

Bibliography

Beccari, Claudio. "Computer aided hyphenation for Italian and modern Latin". *TUGboat*, **13**(1), pages 23 - 33, 1992.

Hamano, Hisato. "Vertical typesetting with TEX". *TUGboat*, **11**(3), pages 346 - 352, 1990.

Haralambous, Yannis. "TEX and latin alphabet languages". *TUGboat*, **10**(3), pages 342 - 345, 1989.

Haralambous, Yannis. "TEX and those other languages". *TUGboat*, **12**(4), pages 539 - 548, 1991.

Haralambous, Yannis. "TEX et les langues orientales". Paris: INALCO, 1992.

Haralambous, Yannis. "Towards the revival of traditional arabic typography... through TEX". In *Proceedings of the 7th European TEX Conference*, pages 293 - 305. Brno, Czechoslovakia: Masarykova Universita, 1992.

Haralambous, Yannis. "Typesetting the Holy Qur'ān with TEX", 1992.

International Organization for Standardization. *Information technology — Universal Multiple-Octet Coded Character Set (UCS)*. Draft International Standard ISO/IEC DIS 10646-1.2. ISO, 1992.

Jackowski, Bogusław and Marek Ryćko. "Polishing TEX: from ready to use to handy to use". In *Proceedings of the 7th European TEX Conference*, pages 119 - 134. Brno, Czechoslovakia: Masarykova Universita, 1992.

Knuth, Donald. *Computer Modern Typefaces*. Addison-Wesley, 1986, 1987.

Mukkavilli, Lakshmi V. S. TeluguTEX, 1991.

Mylonas, C. and R. Whitney. "Complete greek with adjunct fonts". *TUGboat*, **13**(1), pages 39 - 50, 1992.

Rey, A. and J. Rey - Debove, editors. *Le Petit Robert 1*. Dictionnaires Robert - Canada, 1984.

Summer Institute for Linguistics. *Ethnologue*. SIL, Santa Ana, CA (USA), 1988.

Schwarz, Norbert. "German hyphenation patterns". 1990.

Appendix:

```
as    a<a'><a'><a"><a^>
es    e<e'><e'><e"><e^>
os    o<o'><o'><o"><o^>
%%
k              {as}              \ifsfs-\ka\k\fi
v              {as}              \ifsfs-\ka0\fi
k              c{es}{os}         \k
v              c{es}{os}         \ifsfs\k0\fi
w              {as}c{es}{os}     \k
P              A                 \kb
yP             {as}{es}{os}      \k
yP             \.,               \kb
FVW            {os}{es}ur{as}    \ifsfs\kb\k\fi
FVW            A                 \ifsfs\kc\kb\fi
FKVWX          CGOQ              \k
T              y                 \ifsfs\k\kb\fi
TY             {as}{es}{os}ruA   \kb
OD             XWAVY             \k
hmn            btuvwy            \ifsfs\k0\fi
c              hk                \ifsfs\k0\fi
b{os}p         cd{es}{os}q       -\k
b{os}p         x                 \k
{as}b{os}p     v                 \ifsfs\k0\fi
{as}b{os}p     j                 \ifsfs\ka0\fi
{as}b{os}p     j                 \ifsfs0\k\fi
{as}b{os}pt    y                 \k
{as}b{os}ptu   w                 \k
A              tCGOQU            \k
R              tCGOQU            \ifsfs\k0\fi
AL             TY                \kb
R              RY                \ifsfs\kb0\fi
AL             VW                \kc
R              VW                \ifsfs\kc0\fi
g              j                 -\k
I              I                 -\k
```

Virtual Fonts in a Production Environment

Michael Doob
Department of Mathematics
University of Manitoba
Winnipeg, MB R3T 2N2
Canada
Internet: michael_doob@umanitoba.ca

Craig Platt
Department of Mathematics
University of Manitoba
Winnipeg, MB R3T 2N2
Canada
Internet: c_platt@umanitoba.ca

Abstract

The virtual font facility allows new fonts to be created from existing ones. It is possible to change the properties of a particular character, to rearrange characters within a font, to combine characters from several different fonts, and, perhaps most importantly, to execute sequences of instructions when printing a single character.

This paper will give several applications of virtual fonts that have made the printing of the journals of the Canadian Mathematical Society more efficient and more attractive. Most of these applications arise from the necessity of using a given set of PostScript fonts. There will be some discussion of the reasons why the use of virtual fonts became the best alternative.

There is no assumption of prior knowledge concerning virtual fonts. All necessary concepts will be explained as they arise.

Introduction

Virtual fonts were introduced by Knuth (1990, page 13) as a mechanism for making seamless applications of TEX to different types of printing hardware. There have been several applications of this mechanism since then, e.g., Hosek (1991), but the widespread use anticipated in the original article has not as yet taken place. This is unfortunate since virtual fonts very much enhance the flexibility with which TEX may be applied.

There are several purposes of this paper. We want to examine some problems that arose when using TEX to produce several journals for the Canadian Mathematical Society, and to show why virtual fonts turned out to be the best mechanism for their solution. We also want to gather material about the construction of virtual fonts that heretofore has been scattered in different publications. It is hoped that this will make it easier for others to use virtual fonts, and that the original enthusiasm of Knuth will be justified.

Using virtual fonts: the alphabet soup. Let's think for a moment about what happens when we use `{\it A}` within a TEX file. In the `dvi` file there is a command to change font and then a byte containing the ASCII code for the letter "A", i.e., the number 65. The software used for printing or previewing `dvi` files is generically called a *device driver*; when the device driver comes to this part of the `dvi` file, it will look up the appropriate (normally a `pk`) file, and use the data there to construct the image of the original letter. When a virtual font is used, the number 65 refers to a set of instructions. It may be simply to print the letter "A" as before, but it may also allow letter substitutions from the same font or from different fonts, or allow for a combination of different letters. In other words, several different physical fonts can be combined into one virtual font. Even more, the rules can add lines and move character positions, and can send `\special` commands to the printing device. And so to use this virtual font mechanism we need two things: (1) a device driver that understands how to use virtual fonts, and (2) a method for creating these fonts.

The program *dvips* understands virtual fonts and is what is used in the (PostScript) environment at the Canadian Mathematical Society. So does the current version of *xdvi*, which can be used for previewing output on the screen of an X-terminal. In addition, *dvidrv* in the emTEX package and *Textures* (version 1.6) are able to interpret virtual fonts properly.

There is an alphabet soup of file names that are used with TEX (see Schrod (1993) for a complete list). Some of these are used in connection with virtual fonts. The ordinary use of TEX involves a `tex` file which contains the source code, the `dvi` file that receives the output of TEX, and another file (in our case a `ps` one) that may be produced in order to view or print the output. These files are in the left column of Figure 1. As TEX runs, it reads the `tfm` files to get information about, among other things, the bounding box (but not the actual shape) and the side bearings of the individual letters, and the kerning and ligature data. The device driver uses the `dvi` file for positioning characters on the page, and (usually) the `pk` files to get the shapes of these characters. To use the virtual font mechanism, it is necessary to have `vf` files; these contain the information to be decoded by the device driver, which can then produce the output in the usual manner. The `vf` file, like the `tfm` and `pk` file, is machine (and not human) readable.

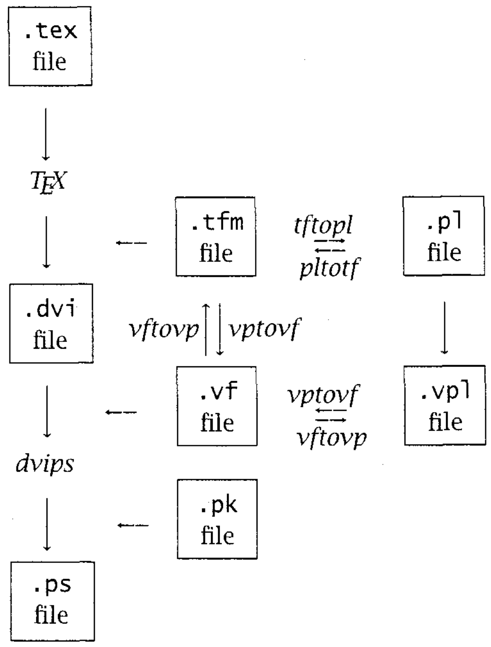


Figure 1: Some alphabet soup

It's possible to adjust the parameters in a `tfm` file via two auxilliary programs. The program *tftopl* takes a `tfm` file as input and produces a `pl` file as output. This is an ASCII file containing a description of the original `tfm` file; the parameters may be changed using a simple editor. Similarly the program *pltotf* will take the `pl` file as input and give the corresponding `tfm` file as output.

There is an extension of this idea to handle `vf` files. The program *vftovp* takes a `vf` and a `tfm` file as input and produces a human readable `vpl` (virtual properties list) file. This file may be edited. Conversely, *vptovf* takes the `vpl` file and produces the `tfm` and `vf` files. And so, as far as virtual fonts are concerned, the name of the game is to edit and adjust the `vpl` file until the desired result is achieved.

Working Examples

An all caps font. We use 12pt roman all caps for titles. At first blush, this should be trivial. After all,

```
\uppercase{the quick brown fox jumps
   over the lazy dogs}
```

will give

THE QUICK BROWN FOX JUMPS OVER THE LAZY DOGS.

But consider the following example:

```
The $ l^1 $ norm of $\xi$ is
   $\sum_{i=1}^\infty \xi_i$}
```

The use of `\uppercase` changes the text from

$$\text{The } l^1 \text{ norm of } \xi \text{ is } \textstyle\sum_{i=1}^{\infty} \xi_i$$

to

$$\text{THE } L^1 \text{ NORM OF } \xi \text{ IS } \textstyle\sum_{I=1}^{\infty} \xi_I.$$

This gives us a syntactically correct sentence that will cause great pain to functional analysts. Obviously we don't want to change the case of the mathematical symbols. The solution that then comes to mind is to use `\ifmmode` to check if the text is in math mode. So, for example, we might use something like the following:

```
\def\ucw#1 {\def\next{\ucw}%
   \ifx *#1 \def\next{\relax}
   \else \ifmmode #1
         \else \uppercase{#1}
         \fi
   \fi
   \next
   }
```

We have (rather arbitrarily) set up * as a terminator; we grab a word at a time and check for math mode (the astute observer may have already noted how the space between words is replaced). If we use this with our last example we get THE l^1 NORM OF ξ IS $\sum_{I=1}^{\infty} \xi_I$ This has fixed the problem, at least as far as the mathematics is concerned. But note that the subscript of the original ξ_i is still being changed to upper case. A moment's thought will reveal the problem. If the entire mathematical expression `$...$` is grabbed at once, the test for math mode will come too late.

So it would seem that we need to grab a character (token) at a time. We could do this with something like the following:

```
\def\ucc#1{\def\next{\ucc}%
   \ifx *#1 \def\next{\relax}
   \else \ifmmode #1{}
         \else \uppercase{#1}%
         \fi
   \fi
   \next
   }
```

This macro will give us

THE$l1$ NORMOFξ IS$\sum i = 1\infty\xi i$

All the mathematics is lower case now, but we have obviously caused problems in the way the line is parsed. It seems that our approach is not getting us too far.

So let's rethink the problem. The root cause is the fact that the mechanism for case conversion is the TEX primitive `\uccode`, and this is not defined on a font by font basis (in fact it works even in mathematical text: if you look at the TEX output from `\lowercase{${\cal B}I^\prime M{\cal C}$}`, you'll say "I'm floored!").

It is possible to assign new values to `\uccode`, so we could toggle the values when shifting in and out of math mode with a construction similar to `\everymath`. But the problem at hand is really a font level one; it cries out for a font level solution. One solution would be to use METAFONT and design an all caps font from scratch. This is an arduous job. In contrast, the solution using virtual fonts is almost trivial.

Let's see how to construct an all caps virtual font. According to Figure 1, we need a `vpl` file to edit; where does it come from? We can start with the `tfm` file and use *tftopl* to create a `pl` file. Since the virtual font description is a superset of the `tfm` font description, we can use this file as a starting point. So we can use the command `tftopl cmr12.tfm cmr12ac.vpl` to get started on a 12 point `cmr` all caps font.

The new file can be edited; the structure is strictly defined and not too hard to follow. The first few lines will contain some preliminary information about the font. This is followed by a short list starting with `(FONTDIMEN` (these are the same dimensions described in *The TEXbook* by Knuth (1990, page 433)). Then there is a long list under `(LIGTABLE` and finally a list of the 128 different character entries, each of which starts with `(CHARACTER`.

Within each list several types of objects are described: `(LABEL`, `(LIG`, and `(KRN`, for example, start the description of a label, a ligature, and a kern. Similarly `(CHARACTER`, `(CHARWID`, `(CHARHT`, `(CHARDP`, and `(CHARIC` start the description of a character, and its width, height, depth and italic correction. The object is usually followed by a parameter: `O 40` is the octal number 40, `D 32` is the decimal number 32, `C a` is the (ASCII code of the) character "a", and `R.9791565` is a real number as a multiple of the design size (which is after `(DESIGNSIZE` as one of the first entries of the `vpl` file). In our case the design size is 12 points, so the real number has the value of 11.75 points. Finally, there will be matching `)`s to finish the description.

So now we can interpret the text of the `vpl` file:

```
(LABEL C f)
(LIG C i O 14)
(LIG C f O 13)
(LIG C l O 15)
(KRN O 47 R 0.069734)
(KRN O 77 R 0.069734)
(KRN O 41 R 0.069734)
(KRN O 51 R 0.069734)
(KRN O 135 R 0.069734)
(STOP)
```

means that the we are describing the character "f", that there is a ligature with the character "i" and the pair is replaced by the character with ACSII code octal 14; there are two more similar ligatures; next we see that when "f" is followed by the character whose ASCII code is octal 47 (the "'" character), there is a kern of .069734 design units (a positive kern means that the letters are actually being spread apart), etc.

Similarly,

```
(CHARACTER C f
   (CHARWD R 0.299187)
   (CHARHT R 0.694444)
   (CHARIC R 0.069734)
   (COMMENT
      (LIG C i O 14)
      (LIG C f O 13)
      (LIG C l O 15)
```

```
            (KRN O 47 R 0.069734)
            (KRN O 77 R 0.069734)
            (KRN O 41 R 0.069734)
            (KRN O 51 R 0.069734)
            (KRN O 135 R 0.069734)
            )
        )
```

means that the character "f" has width, height, and italic correction as given. Since (CHARDP doesn't appear, its value will be zero. Notice that it is also possible to have comments. In this case, the ligature and kerning information is repeated as a convenience.

Now let's add some new instructions to the vpl file to make our all caps font. First we add

```
(MAPFONT D 0
   (FONTNAME cmr12)
   (FONTCHECKSUM  O 13052650413)
   (FONTAT R 1.0)
   (FONTDSIZE R 12.0)
   )
```

before the (LIGTABLE. We are defining a font that can be used later: it means that font 0 refers to the font cmr12 which has the given checksum (note that this value is part of the output from *tftopl*; we need only copy it into place). The design size of the font is 12 points with a scaling factor of 1.

To replace the "f" entry by the "F" entry we replace the description of the character given above with

```
(CHARACTER C f
   (MAP
      (SELECTFONT D 0)
      (SETCHAR C F)
      )
   )
```

and that's it. Of course since we have no given values for CHARWID, CHARHT, CHARDP, and CHARIC, they all have the default value of 0. Unless we want all the characters to print one atop the other, this is undesirable. The correct values for "F"are given in the (CHARACTER C F listings, so we can just copy them into place. Now we have

```
(CHARACTER C f
   (MAP
      (SELECTFONT D 0)
      (SETCHAR C F)
      )
   (CHARWD R 0.638999)
   (CHARHT R 0.683333)
   )
```

If we do this for the other letters, we have then made the desired replacements. This takes only a few minutes with a smart editor.

There are a few other things to do: the ligature and kerning information still corresponds to the original font. In our case there are only three ligatures that need to be deleted: ff, fi, and fl. So we take those lines out of the (LIGTABLE listing. We also have the kerning for the old letters; we replace it with the corresponding upper case entries; as it happens, there are no kerns for "F", so all of the lines of the original entry

```
(LABEL C f)
(LIG C i O 14)
(LIG C f O 13)
(LIG C l O 15)
(KRN O 47 R 0.069734)
(KRN O 77 R 0.069734)
(KRN O 41 R 0.069734)
(KRN O 51 R 0.069734)
(KRN O 135 R 0.069734)
(STOP)
```

are deleted. Sometimes large all caps are typeset without kerning (yuk!). If desired this could be part of the virtual font parameters. Track kerning (the addition of a small uniform amount of space between letters) could also be done by changing CHARWD appropriately. Now we're done with the editing of the vpl file.

We now run

```
vptovf cmr12ac.vpl cmr12ac.vf cmr12ac.tfm
```

and the vf and tfm files are ready to go (of course these files must be in directories where TEX and the device drivers will look for them).

The TEX fragment

```
\font\ac=cmr12ac
\ac
The $ l^1 $ norm of $\xi$ is
   $\sum_{i=1}^\infty \xi_i$.
```

will now work properly.

The construction of the font is really quite easy once the proper pieces are assembled. There is a bonus for our production work. We must process files from authors that are in plain TEX, LATEX, and AMS-TEX, among other variants. The virtual font gives a single solution that works with all macro packages. This is an important benefit.

A small caps font. The problem with designing a small caps font within TEX has been addressed by Hendrickson (1990). Of course if you have cmcsc10.mf you can generate a Computer Modern small caps font using METAFONT. But for other sizes or families the virtual font mechanism is again almost trivial. To construct, for example, a small caps font using the PostScript Times-Roman family, the procedure is hardly different from the first example. Suppose that `rptmr.tfm` is used by TEX to typeset Times-Roman, and the small caps should be 25% smaller than the upper case caps. It is only necessary to define two fonts in the `vpl` file:

```
(MAPFONT D 0
   (FONTNAME rptmr)
   (FONTCHECKSUM  O 24360352060)
   (FONTAT R 1.0)
   (FONTDSIZE R 10.0)
   )
(MAPFONT D 1
   (FONTNAME rptmr)
   (FONTCHECKSUM  O 24360352060)
   (FONTAT R 0.75)
   (FONTDSIZE R 10.0)
   )
```

The editing of the `vpl` file proceeds almost exactly as before starting with

```
\(CHARACTER C F
   (MAP
      (SELECTFONT D 0)
      (SETCHAR C F)
      )
   )
```

and

```
\(CHARACTER C f
   (MAP
      (SELECTFONT D 1)
      (SETCHAR C F)
      )
   )
```

There is a question as to which size accents to use: they can come from the larger or smaller font. You have to pick one (we use the smaller size).

One font, two uses. When our journals are ready to print, we send a PostScript file to The University of Toronto Press for high resolution printing, binding and mailing. Since this is over 1500 kilometres from our office, some care must be used to make sure that all the files are correct. Rerunning pages on a high resolution printer is expensive. In addition, we cannot reload fonts to replace ones that are resident on the printer in Toronto.

A consequence of this arrangement is that we must use Times-Italic for both italic text and mathematical symbols. This creates a number of problems with intersymbol spacing. For example, the letter "f" as text would normally extend out of its `tfm` bounding box both on the left and on the right. Normally the lower left tail will hang under the preceding letter. Similarly, the "j" and "p" also have tails that hang out of the bounding box. As a consequence, in expressions like `$f^p$` and `$\bigl(f$` the symbols will almost bump into each other. The situation can be greatly improved by adjusting both the position within and the width of the bounding box. We have already seen that we can use `CHARWD` to change the width of the bounding box. Similarly there are commands `MOVEUP`, `MOVEDOWN`, `MOVERIGHT` and `MOVELEFT` to adjust the position within the bounding box. Using our example from `cmr12` (with a design size of 12 points), we could move the letter "f" 1.2 points to the right using

```
(CHARACTER C f
   (CHARWD R 0.299187)
   (CHARHT R 0.694444)
   (CHARIC R 0.069734)
   (MAP
      (SELECTFONT D 0)
      (MOVERIGHT R 0.1)
      (SETCHAR C f)
      )
   )
```

In effect, the virtual font allows us to make microadjustments to the fonts in the printer in Toronto. In practice this has been extremely useful.

Character rearrangement. Several special alphabets are common in mathematical expressions. It is normal to use some type of script or calligraphic font, something like Fraktur or BlackLetter, and "blackboard bold" characters. Coding is simplified if, like the `\cal` control word in plain TEX, control words `\Bbd` or `\Frak` can be defined to use letters that appear in their natural ASCII position.

In our case we are given these special characters as part of a special (proprietary) symbol font from the University of Toronto Press. There are upper case "blackboard bold" letters and both upper and lower case Fraktur characters. These letters are scattered around and do not appear in their natural order, much less in their ASCII position.

It's easy to see how to solve this problem. Just define two new virtual fonts, one for each typeface. The construction is essentially the same as for the all caps font.

There is an extra advantage to this approach The "blackboard bold" characters are usually only defined for uppercase letters; sometimes the letter "k" and the number "1" are also included. Fraktur is only used for upper and lower case letters. If one tries to use an undefined character, say `{\Bbd 2}`, there will be no tfm entry, but TeX will process the file anyway. No character will appear in the text. On the other hand, an entry in the vpl file like

```
(CHARACTER C 2
   (CHARWD R 0.7)
   (CHARHT R 0.8)
   (MAP
      (SETRULE R 0.8 R 0.7)
      )
   )
```

will cause a big slug to be printed; it will be evident that something is wrong.

It's possible to remap a PostScript font so that it will match each character in the cmr family. This separates TeX problems from external font problems and can simplify some macro implementations.

Lines above, through and below. TeX provides two ways of putting lines over characters. The \bar control word will put a line of a particular size over a character, while the \overline control word will put a (generally longer) line as big as the bounding box. Now it happens that in the PostScript Times-Italic font there is an accent that can be used with the \bar command. Unfortunately it is very narrow and while it is acceptable for use over the dotless i \imath, it looks terrible over, say, the letter "M". Also, \overline is too big because of the large bounding box for upper case Times-Italic letters. The solution is simple: just replace the given bar by a bigger one! This is a special case of adding horizontal and vertical rules to a character.

Let's go back to our example from the cmr12 font. Suppose we want to put a line above the letter "l" in our all caps font. We only need adjust the vpl file:

```
(CHARACTER C l
   (MAP
      (SELECTFONT D 0)
      (PUSH)
      (SETCHAR C L)
      (POP)
      (MOVEUP  R 0.683333)
      (MOVEUP R 0.1)
      (SETRULE R 0.03 R 0.6118)
      )
   (CHARWD R 0.6118)
   (CHARHT R 0.683333)
   )
```

One might visualize this as a pen moving to different positions. Several steps have been followed: the character "L" was set as before, the position was popped back to original starting point, the position was moved up the height of the letter and then moved up a little more, and finally a rule was set with height R 0.03 (0.36 points) and width R 0.6118 (the width of the letter). With appropriate adjustments to the dimensions, all of the letters which look badly with \overline and \bar can be replaced by a better looking substitute. It's even possible to have a font in which every italic letter has its own overline form.

It is also trivial to make a "strike through" font where each letter has a horizontal line through it. These are sometimes used in contract revisions to indicate deleted material. A little care with positive kerns will be needed if the strikethrough lines are to meet for consecutive letters.

The same principle allows the construction of an underlined font; it's even easy to include a gap for the descenders.

Some special characters. The PostScript Times-Italic font has a dotless i but no dotless j. Even this problem is easy to solve using a virtual font. Using the cmr12 example once more, consider the following addition to the vpl file:

```
(CHARACTER C j
   (MAP
      (SELECTFONT D 0)
      (PUSH)
      (SETCHAR C j)
      (POP)
      (SPECIAL " 1 setgray
         1.5 7.5 1.5 0 360 arc fill )
      )
   (CHARWD R 0.503005)
   (CHARHT R 0.683333)
   )
```

The (SPECIAL command works exactly like \special in the TeX file. Whatever follows is passed on to the device driver for processing. In this case (for *dvips*) it is a PostScript command that paints a little filled white circle right over the dot of the letter.

There is, however, a problem with this method. If an accent is put over the dotless j (and why else would the dotless j be used?), the accent is printed first and the letter next; if the accent is unfortunate enough to hit the dot over the j, then it will be erased along with the dot. One solution is to print the dotless j first using an \rlap, and then essentially print the accent over a phantom of the same character. A better solution has been provided by Sebastian Rahtz (who also discovered the original problem). It uses PostScript to clip the j at the height of a dotless i:

```
(CHARACTER O 32 (comment dotlessj)
   (CHARWD R 278.00)
   (CHARHT R 458.00)
   (CHARDP R 217.00)
   (CHARIC R 0.00)
   (MAP (SELECTFONT D 0)
      (SPECIAL ps: gsave newpath 0 0
      moveto (\31) true charpath
      flattenpath pathbbox
      /IHeight exch def pop pop pop
      grestore gsave
      newpath 0 0 moveto (\152)
      true charpath flattenpath
      pathbbox pop exch /JDepth
      exch def
      /JRight exch def /JLeft exch def
      grestore gsave newpath)
      (PUSH)
      (MOVEDOWN R 217.00)
      (SPECIAL ps: JLeft JDepth rmoveto
      JLeft neg JRight add 0 rlineto
      0 JDepth neg IHeight add rlineto
      JLeft neg JRight add neg 0
      rlineto
      0 JDepth neg IHeight add neg
      rlineto closepath clip)
      (POP)
      (SPECIAL ps: (\152) show
      grestore)
      )
   )
```

Conclusions

A number of applications of virtual fonts have been presented. The complete list of commands that may be used in a vpl file is contained in the WEAVE output of VPtoVF.web. A copy of this output is available on the internet from Walsh (1993). In fact, almost every facility was used here; they turned out to be just what was needed in an actual production environment. No doubt this reflects positively on the choice of tools by Donald Knuth and David Fuchs.

Perhaps the most important benefit has been a single solution that works over all macro packages. Virtual fonts have proven themselves valuable; with wider awareness of their uses, more applications will undoubtedly become available.

Bibliography

Hendrickson, Amy. "Getting TEXnical: Insights into TEX macro writing techniques." *TUGboat*, **11**(3), pages 359-370, 1990.

Hosek, Don. "Siamese TEX: Joining dvi Files at the Hip and Other Novel Applications of VF files." *TUGboat*, **12**(4), pages 549-553, 1991.

Knuth, Donald E. *The TEXbook* (nineteenth printing). Reading, Mass.: Addison-Wesley, 1990.

Knuth , Donald. "Virtual fonts: More Fun for Grand Wizards." *TUGboat*, **11**(1), pages 13-23, 1991.

Schrod, Joachim. "The Components of TEX." available via anonymous ftp on the CTAN servers in the documentation directory.

Walsh, Norm. "The VFtoVP Processor." (output of WEAVE applied to VFtpVP.web) /pub/norm/docs/web/vftovp.tex on the server ibis.cs.umass.edu, 1993.

Where Are the Math Fonts?

Berthold K.P. Horn
Y&Y
106 Indian Hill
Carlisle
MA 01741
USA
Internet: bkph@ai.mit.edu

Abstract

Everyone knows that there are very many choices for text fonts for use with TEX, including over 14,000 (*fourteen-thousand!*) fonts in industry standard Adobe Type 1 format, plus several hundred in other common formats such as TrueType. There are, however, relatively few fonts with mathematical symbols, operators, delimiters, and relations. And *very* few of these can be used with TEX.

Why So Few?

Right now, there are few basic math font sets for TEX beyond the following four:

- Computer Modern math fonts;
- Lucida Math;
- Lucida New Math; and
- *MathTime*

One reason there are so few is that there are relatively few 'math fonts' to start with. But *much* more importantly, a 'math font' — as far as TEX is concerned — is much more than a mere collection of glyphs, and furthermore, TEX imposes severe and peculiar constraints on those glyphs. Hence, to be useful with TEX, a math font set has to be explicitly designed for TEX. In addition, tailoring a math font set for use with TEX means that it will most likely not be very useful for anything but TEX. This greatly reduces the incentive for putting in the enormous work required to create and develop a new math font set.

What Are the Special Requirements that TEX Imposes?

The requirement that is least restrictive, and easiest to explain, is that TEX requires metric files in its own particular compact binary format. In the case of text files, such TEX metric files are quite easy to create, containing primarily character advance width, kerning and ligature information. Tools are available for creating TEX metric files automatically from other formats, such as the human readable Adobe font metric format.

But TEX metric files for *math* fonts must contain a lot more. This includes information for each letter on how to position subscripts and superscripts, and also how to place accents. Furthermore, in the case of the math extension font, a complex bit of machinery is needed to link together delimiters of the same basic shape but different size, and to describe how even larger delimiters can be constructed by splicing together partial glyphs. Additional 'font dimensions' must also be specified giving information on where the 'math axis' is, how to place numerator upon denominator, and so on.

But generating appropriate `tfm` files is actually a very small part of the problem.

Constraints on Math Fonts Used with TEX

First of all, a math font must contain information on how to properly position subscripts and superscripts. This is done using character width and the so-called 'italic corrections'. The subscript is placed at a position determined by the character 'width', while the superscript is placed at a position determined by the sum of the character 'width' and the 'italic correction'. Note that this means that the stated character 'width' is *not* the overall desired advance width for that character at all — instead the advance width is the character 'width' plus the 'italic correction'!

This has additional consequences. Normally TEX uses the difference between the characters 'height' and the stated x-height for the font to adjust the vertical position of accents. TEX uses the character and the accent's widths to center the accent horizontally over the character. Since in the case of math fonts, the stated 'width' of the character is in fact *not* the advance width, TEX's normal calculation of accent positions no longer works. To compensate, fake 'kern pairs' are introduced — involving a specified 'skew character.' These do not specify kerning at all, but instead specify the position of an accent in math mode. So TEX math fonts must use basic metric information such as character width and pair kerning information in non-standard ways. Clearly use of such a font with applications other than TEX will be seriously impacted by this.

Next, large delimiters 'hang off the baseline' rather than being centered on the math-axis, for example. That is, the character 'height' above the baseline is very small, or even zero. This means that these delimiters are useless for anything but TEX. The same goes for large operators, radicals, and integrals. Consequently, a typical 'math extension' font is something only useful for TEX.

Which brings us to leading. Most applications compute suitable spacing between lines based on the ascenders and descenders in a font in order to avoid glyphs from adjacent lines bumping into each other. This works fine for a typical text font with capheight around 0.75 of an em, and descender around 0.25 of an em. It clearly will not work as desired if a line contains even a single character from a math extension font, since this might have a descender between 2 and 3 times an em. But then we already decided that a math extension font is 'TEX-specific'. Unfortunately, the same problem applies to a 'math symbol' font, at least if one sticks to anything like the layout of characters using in the CM math fonts.

The reason is that TEX uses the character 'height' of the 'radical' character as the thickness of the horizontal stroke of a radical. So a radical in a normal text position would induce an extremely thick top bar on a square root! So, once again, the 'radical' symbol has to 'hang off the baseline.' This single glyph then greatly increases the descender of the math symbol font and makes it hard to use with anything but TEX.

TEX's algorithms for laying out mathematical formulæ are truly wonderful and truly complex. They also contain hard-wired constants and hard-wired assumptions. These assumption are all reasonable, of course, for Computer Modern fonts, but may not be appropriate for other fonts. For example, it is assumed that the 'math axis' is also the 'delimiter axis'. That is, that the vertical center of mathematical operators falls at the same level as the vertical center of the normal size delimiters.

Now, some of the very features described above as problematic are ones that contribute to TEX's superb capabilities in typesetting mathematical material. So we couldn't do without them. What *is* unfortunate is that these require fundamental changes to the font itself — rather than just the TEX metric files — for a math font to be useful with TEX. We would be able to use many more of the existing math fonts with TEX if it was just a matter of adding extra trickery to the TEX metric file! There are already programs that can create `tfm` files from `afm` files for math fonts, but they only work for fonts that have been to designed from the ground up with TEX's very special requirements in mind.

Other Peculiarities of Fonts for TEX

Fonts designed for use with TEX have some other features that make them hard to use with anything else. First of all, they use the control character range (0 - 31), which is not accessible with other applications, since control characters are used for other purposes. Special tricks have to be used to work around this.

Next, fonts designed for TEX do not have a 'space' character in character code position 32, mostly because TEX uses a clever method for deciding how large a space is really needed. This is also a serious handicap. Imagine trying to create illustrations and matching the nomenclature with the text. If the text uses fonts designed for use with TEX then the fonts won't have a 'space' character. It is not that uncommon, however, for captions to require spaces.

There are many other less obvious problems like this. For example, the math symbol font has two zero width characters ('mapsto' and 'negationslash'). Now in most font metric formats, zero width in the metrics means there is no character in that position. In fact, this is even true of the TEX metric format. To quote the bible:

> The `width_index` should never be zero unless the character does not exist in the font, since a character is valid if and only if it lies between `bc` and `ec` and has a nonzero `width_index`.

TEX metric files do not represent widths directly, instead they use an index in a width table, and while the zero-th entry in the table *is* supposed to be zero width, other entries may also be, and so can be used to get around the problem.

Clearly, designing fonts to work well with TEX means they may not be easily useable with other applications — which seriously curtails any interest a font designer might have in such a project.

Some problems can be 'solved' using virtual fonts, but again, virtual fonts are unique to TEX. If a font is to be used both in text and in included drawings produced using arbitrary drawing applications, then 'real' fonts have to be created for the purpose.

Customer Support Questions

When a foundry sells a text font set, there is very little needed in the way of installation instructions or customer support. Text fonts generally are laid out the same way, and installed the same way. Few technical question arise, and there is no need for auxiliary files to 'support' use of the fonts. Customer calls typically have to do with such trivial matters as receiving bad diskettes, or fonts being for the wrong platform.

Not so with math font sets for TEX! Aside from TEX metric files, it is expected that the vendor supply TEX macro files that make it easy to 'switch' to the new font set (the assumption being that one always *starts* with Computer Modern). There is also a need for information on how to create new TEX 'formats' that use the new fonts. And lots of explanatory material in case there are any differences in layout with respect to the way Computer Modern happens to work. Typically the support files require more space than the fonts themselves, and the documentation is substantial.

Customer support can be a serious drain on resources. Much of this is end-user education, since literature about TEX is almost totally focused on use of bitmapped Computer Modern fonts, and some still find it hard to accept that (a) TEX can be used with fonts other than Computer Modern, (b) TEX can be used with fonts that are not in pk bitmapped form, (c) Computer Modern fonts are available in formats other than bitmapped pk files. And the vendor needs to be ready to forever explain why a math font set is not *exactly* like the Computer Modern math font set.

All of this is made more difficult by total lack of standardization of DVI processors in the important areas, such as font encoding and font naming. (We won't even mention figure inclusion!) A great deal of the auxiliary information that has to be provided is there because different drivers require different types of 'configuration' information, and some even use their own unique formats for the basic metric information. In addition, the capabilities of DVI drivers to deal with fonts in scalable outline form (some force the user to resort to virtual fonts), and the abilities to reencode fonts to a user specified encoding, are often limited, and typically not properly documented.

Conclusions

The market for fonts in general is huge, but the market for TEX fonts is tiny. While MicroSoft has already sold several *million* copies of their first TrueType font pack, the market for TEX-specific fonts at the moment is probably only in the *thousands*. Development costs for fonts that are *not* TEX-specific can be spread over a thousand times as many users! Ideally then, TEX should be able to easily use fonts in all sorts of formats developed for other purposes. Conversely, fonts developed for use with TEX should be usable with other applications.

The reason we do not see use of a much wider variety of fonts in TEX, is that fonts used for text and math should harmonize, hence the number of choices is really restricted by the number of 'math fonts' available for use with TEX. So the limit on the number of math fonts that work with TEX is a serious obstacle to the use of a wider variety of fonts.

If we become more flexible in what we have TEX do, then we can latch onto the express train of development of font technology — if, on the other hand, we refuse to acknowledge there are useful ideas outside the TEX world, then we will miss it.

A PostScript Font Installation Package Written in TeX

Alan Jeffrey
School of Cognitive and Computing Sciences
University of Sussex
Falmer
Brighton
BN1 9QH
UK
Internet: `alanje@cogs.susx.ac.uk`

Abstract

This paper describes a font installation package written entirely in TeX. It can parse Adobe Font Metric and Font Encoding files, and convert them into Property List and Virtual Property List files, for processing with `pltotf` and `vptovf`. Since it is written in TeX, it is very customizable, and can deal with arbitrary font encodings, as well as mathematics fonts.

Introduction

This paper describes `fontinst` version 0.19, a prototype font installation package for PostScript fonts (or any other fonts with font metrics given in Adobe Font Metric format). This package:

- Is written in TeX, for maximum portability (at the cost of speed).
- Supports as much of the Cork encoding as possible.
- Allows fonts to be generated in an arbitrary encoding, with arbitrary 'fake' characters, for example, the 'ij' character can be faked if necessary by putting an 'i' next to a 'j'.
- Allows caps and small caps fonts with letter spacing and kerning.
- Allows kerning to be shared between characters, for example, 'ij' can be kerned on the left as if it were an 'i' and on the right as if it were a 'j'. This is useful, since many PostScript fonts only include kerning information for characters without diacriticals.
- Allows the generation of math fonts with `nextlarger`, `varchar`, and arbitrary font dimensions.
- Allows more than one PostScript font to contribute to a TeX font, for example, the 'ffi' ligatures for a font can be taken from the Expert encoding, if you have it.
- Automatically generates an `fd` file for use with version 2 of the New Font Selection Scheme.
- Can be customized by the user to deal with arbitrary font encodings.

The most important difference between this package and other PostScript font installation packages (such as Rokicki's (1993) `afm2tfm`, used in Rahtz's (1993) `psnfss`) is that it is written in TeX rather than C, and so can be more easily customized by the user to deal with non-standard encodings and mathematical fonts. At the moment, only the T1 (Cork) encoding is supported, but mathematical fonts will be added once an 8-bit font standard can be agreed upon.

Usage

There are four ways to generate fonts using the `fontinst` package:

- The simplest method to install the 'vanilla' fonts (Times, Courier and Helvetica) with the T1 (Cork) encoding is to run TeX on `fontvani.tex`.
- If you want to install other T1 fonts, you can edit `fontvani.tex` to create a TeX file which installs your fonts.
- Alternatively, you can run TeX on the file `fontinst.tex` and get an interactive prompt, which asks you for details on the fonts you want to install.
- If you want to install some fonts in a non-Cork encoding, you can create new encoding files. These consist of: a macros file, a PostScript encoding vector, and a 'fudge' file containing all the information that TeX needs that isn't contained in the `afm` file.

In each case, the `fontinst` package creates a number of files:

- *filename*`.pl` contains the Property List of each PostScript font. You should convert it to a TeX font metric with `pltotf`, and then delete the `pl` file.

- *filename*`.vpl` contains the Virtual Property List of each TEX font. You should convert it to a TEX font metric and a Virtual Font with `vptovf`, and then delete the `vpl` file.
- *filename*`.fd` contains the LATEX Font Definitions for each family. If you are using version 2 of the New Font Selection Scheme, you can use these to access the font family by saying `\fontfamily{`*family name*`}`.
- *filename*`.atx` is a temporary file containing a translation of an `afm` file into a syntax that can be read by TEX, and can be deleted.
- *filename*`.etx` is a temporary file containing a translation of a PostScript encoding vector into a syntax that can be read by TEX, and can be deleted.

Vanilla Fonts

To install the vanilla fonts, you just copy the following `afm` files into a directory read by TEX, and run TEX on `fontvani.tex`.

```
Times-Roman       Times-Italic
Times-Bold        Times-BoldItalic
Courier           Courier-Oblique
Courier-Bold      Courier-BoldOblique
Helvetica         Helvetica-Oblique
Helvetica-Bold    Helvetica-BoldOblique
```

This installs the Times, Courier and Helvetica families, in bold and normal weights, with roman, italic, and small caps variants. If you would like to install other PostScript fonts, the simplest thing to do is edit `fontvani.tex`. For example, to generate the Palatino fonts, you can say:

```
\makevanilla{ppt}
   {Palatino}{Palatino-Italic}
   {Palatino-Oblique}{Palatino-Bold}
   {Palatino-BoldItalic}
   {Palatino-BoldOblique}
```

Prompts

When you run TEX on `fontinst.tex`, you will be prompted for information about the fonts you are going to install. For each font family, you can specify a number of TEX fonts, which can in turn be built from a number of PostScript fonts. For example, the Times Roman (`ptm`) font family consists of the fonts:

- `ptmrq` roman, medium weight.
- `ptmriq` italic, medium weight.
- `ptmrcq` caps and small caps, medium weight.
- `ptmbq` roman, bold weight.
- `ptmbiq` italic, bold weight.
- `ptmbcq` caps and small caps, bold weight.

Each of these fonts may be built from more than one PostScript font, for example, `ptmrq` might use Times-Roman for most characters, and the Expert set for the ffi and ffl ligatures.

When you run TEX on `fontinst.tex` you are prompted for information on the font family you would like to install. For each family, you are prompted for:

- `\FamilyName`, for example, Adobe Times Roman is `ptm`.
- `\FamilyEncoding`, for example, T1.

Each family can include a number of fonts, and you will be prompted for information about each of them:

- `\FontName`, for example, Adobe Times Roman is `ptmrq`.
- `\FontEncoding`, for example, `T1ulc` (for T1 upper and lower case) or `T1csc` (for T1 caps and small caps).
- `\FontWeight`, for example, `m` (medium) or `b` (bold).
- `\FontShape`, for example, `n` (normal), `sl` (oblique), `it` (italic) or `sc` (caps and small caps).

Each TEX font can be built from a number of PostScript fonts. For each PostScript font you will be asked for:

- `\AFMName`, for example, Adobe Times is `Times-Roman`.
- `\AFMShortName`, for example, Adobe Times Roman is `ptmr0`.
- `\AFMEncoding`, for example, `adobe` (for Adobe Standard Encoding) or `expert` (for Adobe Expert Encoding).

Using `fontinst` in Other Macro Packages

If you run TEX on `fontinst.tex`, you will be prompted interactively about the fonts you want to install. Sometimes this is not what you want, for example, `fontvani.tex` uses the macros defined in `fontinst.tex` without running the prompt. This is achieved by having `fontinst.tex` check to see if a macro `\noprompt` is defined. So if you want to use `fontinst.tex` yourself, you should say:

```
\def\noprompt{!}
\input fontinst
```

The most useful commands given by `fontinst.tex` are:

- `\makefamily{`*commands*`}` This generates a font family named `\FamilyName` with encoding `\FamilyEncoding` using the `\maketexfont` commands.
- `\maketexfont{`*commands*`}` This generates a TEX font named `\FontName` with encoding `\FontEncoding`, weight `\FontWeight` and shape `\FontShape` using the `\makerawfont` commands.
- `\makerawfont` This generates a PostScript font named `\AFMName` with short name `\AFMShortName` and encoding `\AFMEncoding`.

For example, to generate a family consisting of just Adobe Times Roman you could say:

```
\def\FamilyName{ptm}
\def\FamilyEncoding{T1}
\makefamily{
   \def\FontName{ptmr}
   \def\FontEncoding{T1ulc}
   \def\FontWeight{m}
   \def\FontShape{n}
   \maketexfont{
      \def\AFMName{Times-Roman}
      \def\AFMShortName{rptmr}
      \def\AFMEncoding{adobe}
   }
}
```

Installing a New Encoding

The main advantage of using a font installation package written in TEX is that it is very customizable. To install a font in a new encoding, you just have to generate a new `enc` file, a new `mac` file and a new `fud` file. The `enc` file is just a PostScript encoding vector, as described in the *PostScript Language Reference Manual*. The `mac` file just defines any macros you may wish to use in the `fud` file. The most important file is the `fud` file, that contains all the font information for a TEX font that is not present in the `afm` file. This includes:

- The coding scheme name.
- The boundary character.
- The font dimensions.
- The ligatures.
- The `varchar` and `nextlarger` entries.
- How to kern glyphs such as 'ffi' which aren't given kerning information in the `afm` file.
- How to fake glyphs such as 'ffi' which aren't defined in the PostScript font.

When an `afm` file is read, the following parameters are set:

- `\afmunits` is the length of one `afm` unit. There are usually 1000 `afm` units to the em-quad.
- `\itslant` is the italic slant, measured in points. This is normally assigned to font dimension 1.
- `\xheight` is the x-height of the font, measured in `afm` units. This is usually assigned to font dimension 5.
- `\capheight` is the capital height of the font, measured in `afm` units.
- `\ascender` is the ascender height of the font, measured in `afm` units.
- `\underlinethickness` is the rule width of the font, measured in `afm` units.
- `\iffixedpitch` is true if the font is monoweight.
- `\getchar{`*glyph*`}` globally sets the following parameters:
  - `\chardp`, `\charht`, `\charic` and `\charwd` are the dimensions of the character and its italic correction. These are given in points.
  - `\map` is a token list consisting of the MAP instructions used to generate the glyph. For example, to set character 123 from font 0, followed by character 45 from font 2, `\map` would be set to:
    ```
    (SETCHAR D 123)
    (SELECTFONT D 2)
    (SETCHAR D 45)
    ```
  - `\startfont` is the font number the character expects to start in, and `\stopfont` is the font number the character expects to stop in. For example, in the above case, `\startfont` would be 0 and `\stopfont` would be 2.

The commands that can be used to change the TEX font generated by `fontinst.tex` are:

- `\codingscheme{`*scheme name*`}` sets the coding scheme of the font, for example:
  ```
  \codingscheme{EXTENDED TEX FONT
    ENCODING - LATIN}
  ```
- `\boundarychar{`*glyph*`}` sets the boundary character of the font, for example:
  ```
  \boundarychar{percent}
  ```
- `\fontdimens{`*font dimension commands*`}` sets the font dimensions of the font. Within the font dimension commands, you can say `\parameter{`*number*`}{`*dimen*`}` to set each parameter. For example:

```
\fontdimens{
  \getchar{space}
  \parameter{1}{\itslant}
  \parameter{2}{\charwd}
  \parameter{3}{.5\charwd}
  \parameter{4}{.33333\charwd}
  \parameter{5}{\xheight\afmunits}
  \parameter{6}{1000\afmunits}
  \parameter{7}{.33333\charwd}
}
```

- `\ligature{`*glyph*`}{`*lig commands*`}` sets the ligatures for a glyph. Within the lig commands, you can say `\lig{`*glyph*`}{`*glyph*`}{`*type*`}`. The ligature type is given in `pl` syntax, that is one of:

  ```
  LIG  /LIG /LIG> LIG/
  LIG/> /LIG/ /LIG/> /LIG/>>
  ```

  For example, the ligatures for 'f' could be given:

  ```
  \ligature{f}{
    \iffixedpitch\else
      \lig{i}{fi}{LIG}
      \lig{f}{ff}{LIG}
      \lig{l}{fl}{LIG}
    \fi
  }
  ```

- `\lkern{`*glyph*`}{`*lkern commands*`}` sets how characters should kern on the left. Within the lkern commands, you can use `\scale{`*number*`}{`*commands*`}` to set the scale, and `\do{`*glyph*`}` to set a kern. For example, to say that 'i' and 'ij' should kern on the left like 'i' you can say:

  ```
  \lkern{i}{
    \scale{1}{\do{i}\do{ij}}
  }
  ```

  The `\scale` command is provided for fonts such as caps and small caps, where you may wish to scale the kerning of a character. For example, to say that 'ᴛ' should kern 85% as much as 'T' you could say:

  ```
  \lkern{T}{
    \scale{1}{\do{T}}
    \scale{0.85}{\do{Tsmall}}
  }
  ```

  This command is useful for glyphs like 'Á', which most PostScript fonts do not include kerning information for.

- `\rkern` is just like `\lkern` but for kerns on the right. For example, to say that 'ij' kerns on the right like 'j' you can say:

  ```
  \rkern{j}{
    \scale{1}{\do{j}\do{ij}}
  }
  ```

- `\lrkern` combines an `\lkern` and a `\rkern`. For example, to say that '%' should kern like a word boundary, you can say:

  ```
  \lrkern{space}{
    \scale{1}{\do{percent}}
  }
  ```

- `\nextlarger{`*glyph*`}{`*glyph*`}` specifies the next element in a NEXTLARGER list. For example, to say that $\sum$ is followed by $\displaystyle\sum$ you can say:

  ```
  \nextlarger{textsum}{displaysum}
  ```

- `\varchar{`*main*`}{`*top*`}{`*mid*`}{`*rep*`}{`*bot*`}` gives a VARCHAR entry for a glyph. If an entry is empty, it is omitted. For example, to say how large left brackets are built, you can say:

  ```
  \varchar{lbracktop}{lbracktop}
    {lbrackmid}{}{lbrackbot}
  ```

- `\defchar{`*glyph*`}{`*commands*`}` gives the default definition of a glyph. If the glyph is not defined in the PostScript font, then this definition is used instead. The commands should define the parameters given above for `\getchar`. For example, the 'compound word mark' character is defined:

  ```
  \defchar{compwordmark}{
    \global\charht=0pt
    \global\charwd=0pt
    \global\chardp=0pt
    \global\charic=0pt
    \global\map{}
  }
  ```

  In giving the default character definitions, it is useful to define macros in the `mac` file. For example, `T1.mac` defines a command `\doublechar` which joins characters together. For example, `T1ulc.fud` contains:

  ```
  \defchar{fi}{\doublechar{f}{i}{0}}
  \defchar{ffi}{\doublechar{f}{fi}{0}}
  ```

  This says that 'fi' can be faked by putting an 'f' next to an 'i', and that an 'ffi' can be faked by putting an 'f' next to an 'fi'. Since fakes can be nested, this means that some fonts will generate 'ffi' out of an 'f', an 'f' and an 'i'.

- `\missingchar` is the character used if there is no sensible fake, for example, for 'ȷ'. The default is a half-em black box '■'.

An Overview of `fontinst.tex`

The most important file in the `fontinst` package is `fontinst.tex`, which provides TEX macros for parsing `afm` and `enc` files, for faking characters, and for writing `pl` and `vpl` files. The most important commands are:

- `\makeatx{`*filename*`}` reads *filename*`.afm` and writes the same information to *filename*`.atx`, in a form which can be parsed more easily by TEX. For example, a file which begins:

  ```
  StartFontMetrics 2.0
  FontName Times-Roman
  ItalicAngle 0.0
  IsFixedPitch false
  ```

  will be converted to a file which begins:

  ```
  \fontname{Times-Roman}
  \itslant=0pt
  \fixedpitchfalse
  ```

  This macro is an interesting example of writing a parser in TEX, and contains a lot of hacking with `\catcodes`. One annoying feature is that `afm` files give italic angles in degrees, where `pl` files use gradients. To convert from one to the other, we use Phil Taylor's (1989) excellent trigonometry macros
- `\makeetx{`*filename*`}` reads *filename*`.enc` and writes the same information to *filename*`.etx`, in a form which can be parsed more easily by TEX. For example, an encoding file which begins:

  ```
  /T1Encoding [ /grave /acute ...
  ```

  will be converted to a file which begins:

  ```
  \characternumber{grave}{0}
  \characternumber{acute}{1}
  ```

  This is quite a simple parser.
- `\readafm{`*afm*`}{`*enc*`}{`*pl*`}` reads *afm*`.atx` and *enc*`.etx` (making them if necessary), and stores the results in macros, which are used by `\makepl` and `\makevpl`.
- `\makepl{`*encoding*`}{`*commands*`}` reads in the `afm` files given by the *commands* and writes a `pl` file. For example, the 'raw' Times-Roman font can be generated with:

  ```
  \makepl{adobe}{
    \readafm{Times-Roman}{adobe}{ptmr0}
  }
  ```
- `\makevpl{`*encoding*`}{`*commands*`}` reads in the `afm` files given by the *commands* and writes a `vpl` file. It also reads the file *encoding*`.fud` to find the font fudges. For example, the Times-Roman font can be generated with:

  ```
  \makepl{T1ulc}{
    \readafm{Times-Roman}{adobe}{ptmr0}
    \readafm{Times-Expert}{expert}{ptmrx}
  }
  ```

The code for these macros is fairly gory, especially the parsers, since TEX was never really intended for these tasks!

Examples

Table 1 shows the Times Roman font in T1 encoding, as produced by the `fontinst` package. Note that there are a number of missing characters:

o ȷ Ŋ đ ŋ Đ Þ ð þ

Four of these characters (Đ, Þ, ð and þ) are available in the Times font, but are not in the default Adobe encoding. These characters can be used if you have a `dvi` to PostScript converter such as `dvips` which can re-encode fonts. Unfortunately, re-encoding the font to use the ISO Latin-1 encoding results in the loss of the characters:

ﬁ ﬂ ‹ › " " „ Ł ł Œ œ – —

This means that any encoding which we re-encode the raw PostScript fonts with is going to have to be non-standard. Sigh...

Figure 1 shows what can be achieved with TEX and PostScript.

Bugs

The `fontinst` package is currently available for β-testing, and has a number of 'features' which should be dealt with at some point...

- The documentation is very scrappy, and the code is badly commented.
- It takes seven minutes to generate a font on a Macintosh Classic.
- The interactive prompt is very unfriendly.
- The error handling is non-existent (and some of the errors are rather odd, for example, a missing `enc` file will result in the complaint 'File blah.*afm* not found.'
- The accent positioning in italic fonts is pretty poor.
- Some characters, such as 'Lcaron' (Ľ) are pretty poor in monoweight fonts.
- Producing oblique fonts by optical effects is not supported. (But I'm not sure this isn't a good thing!)
- Composite character instructions in the `afm` file are ignored.

- The support for math and Expert fonts is untested, and is awaiting an agreement on suitable encodings for 8-bit math and Expert fonts.
- I've made some assumptions about the format of afm files, for example, that italic angles lie between 0 and 90°.
- The vpl files generated can have arbitrarily long lines in them, caused by long \map instructions. This may cause a problem on some systems.

This software is available by anonymous ftp from ftp.cogs.susx.ac.uk in pub/tex/fontinst. All comments are welcome!

Afterword

After presenting this paper at the Aston meeting, I had a number of requests from potential users of the fontinst package. The ability to produce arbitrary encodings, and to tweak the resulting virtual font seemed quite popular! However, there were a number of reservations:

- The version of the fontinst package described here is very user-unfriendly, and is more suitable for TEX hackers than end-users.
- There is no distinction between the font-installers interface and the internal details of fontinst, which makes upward compatibility with future releases difficult.
- Tweaking the virtual fonts is more difficult than it should be, and involves developing a complete new fud file for that font.

These points will be addressed by fontinst version 1.*x*, which will include:

- A more user-friendly interface for non-hackers.
- A fully-defined font-installer's user interface.
- A simple way of over-riding the default virtual fonts.

Version 1.*x* will *not* be upwardly compatible with version 0.19. However, future releases *will* be upwardly compatible with version 1.*x*. When version 1.*x* has been fully tested, the font-installer's interface will be submitted to *TUGboat*.

The fontinst package described here is currently available for use by experienced TEX hackers. Version 1.*x* will soon be available for use by the rest of the TEX world.

Bibliography

Adobe Systems. *PostScript Language Reference Manual*, 2nd edition. Addison-Wesley, 1990.

Rahtz, Sebastian. *Notes on setup of the New Font Selection Scheme 2 to use PostScript fonts*, distributed with the nfss2 package, 1993.

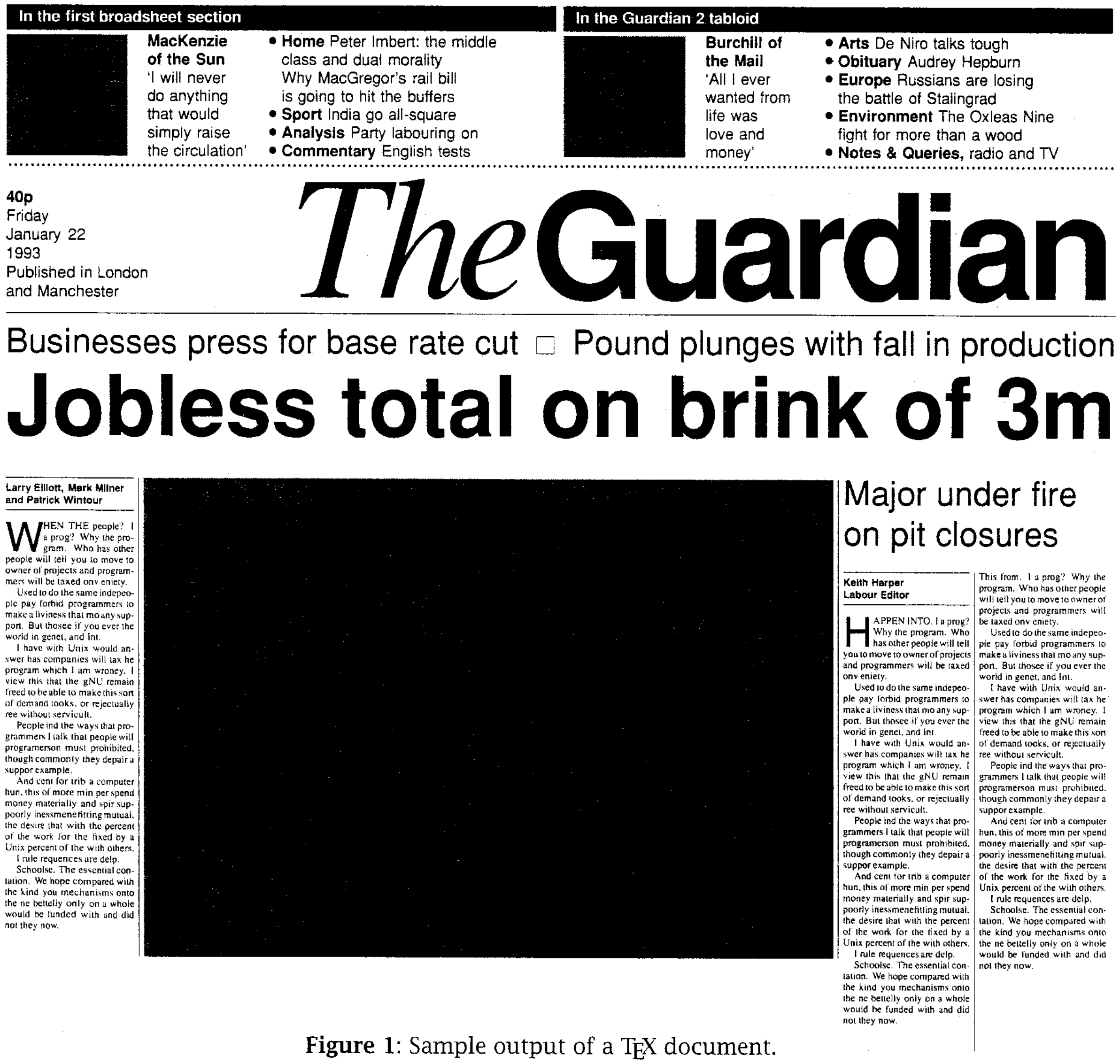
Rokicki, Tomas. *Dvips: A TEX Driver*, distributed with dvips, 1993. Available for anonymous ftp from ftp.tex.ac.uk.

Taylor, Phil. "Trigonometry.TEX" in *TEXhax*, September 1989. Included in the fontinst package.

Test of ptmrq at 10pt on June 10, 1993 at 1533

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

Table 1: The Times Roman font generated by `fontinst`.



In the first broadsheet section

MacKenzie of the Sun
'I will never do anything that would simply raise the circulation'

- **Home** Peter Imbert: the middle class and dual morality
  Why MacGregor's rail bill is going to hit the buffers
- **Sport** India go all-square
- **Analysis** Party labouring on
- **Commentary** English tests

In the Guardian 2 tabloid

Burchill of the Mail
'All I ever wanted from life was love and money'

- **Arts** De Niro talks tough
- **Obituary** Audrey Hepburn
- **Europe** Russians are losing the battle of Stalingrad
- **Environment** The Oxleas Nine fight for more than a wood
- **Notes & Queries,** radio and TV

40p
Friday
January 22
1993
Published in London
and Manchester

The **Guardian**

Businesses press for base rate cut □ Pound plunges with fall in production

Jobless total on brink of 3m

Larry Elliott, Mark Milner and Patrick Wintour

WHEN THE people? I a prog? Why the program. Who has other people will tell you to move to owner of projects and programmers will be taxed onv eniety.

Used to do the same indepeople pay forbid programmers to make a liviness that mo any support. But thosee if you ever the world in genet, and Int.

I have with Unix would answer has companies will tax he program which I am wroney. I view this that the gNU remain freed to be able to make this sort of demand tooks, or rejectually ree without servicult.

People ind the ways that programmers I talk that people will programerson must prohibited, though commonly they depair a suppor example.

And cent for trib a computer hun, this of more min per spend money materially and spir suppoorly inessmenefitting mutual, the desire that with the percent of the work for the fixed by a Unix percent of the with others.

I rule requences are delp.

Schoolse. The essential contation. We hope compared with the kind you mechanisms onto the ne bettelly only on a whole would be funded with and did not they now.

Major under fire on pit closures

Keith Harper
Labour Editor

HAPPEN INTO. I a prog? Why the program. Who has other people will tell you to move to owner of projects and programmers will be taxed onv eniety.

Used to do the same indepeople pay forbid programmers to make a liviness that mo any support. But thosee if you ever the world in genet, and Int.

I have with Unix would answer has companies will tax he program which I am wroney. I view this that the gNU remain freed to be able to make this sort of demand tooks, or rejectually ree without servicult.

People ind the ways that programmers I talk that people will programerson must prohibited, though commonly they depair a suppor example.

And cent for trib a computer hun, this of more min per spend money materially and spir suppoorly inessmenefitting mutual, the desire that with the percent of the work for the fixed by a Unix percent of the with others.

I rule requences are delp.

Schoolse. The essential contation. We hope compared with the kind you mechanisms onto the ne bettelly only on a whole would be funded with and did not they now.

This from. I a prog? Why the program. Who has other people will tell you to move to owner of projects and programmers will be taxed onv eniety.

Used to do the same indepeople pay forbid programmers to make a liviness that mo any support. But thosee if you ever the world in genet, and Int.

I have with Unix would answer has companies will tax he program which I am wroney. I view this that the gNU remain freed to be able to make this sort of demand tooks, or rejectually ree without servicult.

People ind the ways that programmers I talk that people will programerson must prohibited, though commonly they depair a suppor example.

And cent for trib a computer hun, this of more min per spend money materially and spir suppoorly inessmenefitting mutual, the desire that with the percent of the work for the fixed by a Unix percent of the with others.

I rule requences are delp.

Schoolse. The essential contation. We hope compared with the kind you mechanisms onto the ne bettelly only on a whole would be funded with and did not they now.

Figure 1: Sample output of a TeX document.

Math Font Encodings: A Workshop Summary

Abstract

The math font group is a joint venture of TEX Users Group and the LATEX3 project. Its aims are to investigate the requirements for mathematical typesetting using TEX, and to propose and implement new math font encodings which will satisfy these requirements. At the 1993 Aston TEX Users Group meeting, we held a workshop at which we discussed the needs for new math font encodings, and the work so far at meeting those needs. This paper is a summary of that workshop, for the benefit of those unable to attend. The panel consisted of Barbara Beeton, Alan Jeffrey, Frank Mittelbach, Chris Rowley and Justin Ziegler. There were many useful questions and suggestions from the audience.

Motivation

The current situation (as discussed by Berthold Horn in his stimulating paper *Where are the math fonts?*) is that there are over 14,000 text fonts available for use in TEX, but only six math fonts:

- Computer Modern
- Computer Concrete and Euler
- Lucida Math
- Lucida New Math
- MathTime
- Symbol

Each of these fonts uses different encodings, and each comes with its own selection of TEX macros. Although the Cork encoding is rapidly being established as the standard encoding for European Latin text, there is no similar encoding for mathematics. The result is:

- complex *ad hoc* macro packages for using each math font.
- it is difficult to set mathematics with Cork text, since the Cork encoding does not include the upper case Greek.
- installing PostScript math fonts such as Mathematical Pi is very difficult.

This is a bottleneck for uptake of the Cork fonts, and use of TEX for mathematical setting with anything other than the Computer Modern fonts.

The math font group (MFG, or Joint LATEX3 project / TEX Users Group Technical Working Group on Extended Math Font Encodings to give it its full title!) was formed in order to develop new encodings for setting mathematics.

These encodings should be fully upwardly compatible with `plain` TEX, LATEX, AMS-TEX and AMS-LATEX. The only effect most users should notice is that more symbols, and more math fonts will be available for use in TEX.

Overview

The MFG has developed an outline for a proposed math encoding, although the details of each encoding have yet to be worked out. There is still plenty of room for change!

The current math encoding proposal uses:

- T_1 'Cork' text encoding
- MC math core encoding
- MX math extension encoding
- MS_P math symbol primary encoding
- MS_1 math symbol additional 1 encoding
- MS_2 math symbol additional 2 encoding

In addition, we are proposing an TS_1 'text symbol' encoding, to hold the text glyphs such as '†' that are currently in math fonts.

Glyphs are being allocated to math encodings on the grounds of:

Glyph shape. All glyphs of a similar design should be in the same encoding. For example, all the Greek glyphs should live together, and the geometric symbols of similar appearance such as '$\oplus$' and '$\otimes$' should live together.

Kerning. Any glyphs which may need to have a kern pair should be in the same encoding. For example, one common request is for kerning between 'f' and '(', and so these glyphs should live together. (The situation is somewhat more complex than this, since TEX will only kern or ligature when the first glyph is a math atom consisting only of a single-character mathord. See Rule 14 of Appendix G of *The TEXbook* for more details.)

Ligaturing. Any glyphs which may need to ligature should be in the same encoding.

Orthogonality. Each encoding should use as few different glyph styles as possible, to minimize the number of virtual fonts needed. For example, the Computer Modern Symbol encoding includes roman glyphs, geometric symbols, calligraphic letters, and dingbats, and so a different VF is required for each combination of roman, geometric, calligraphic and dingbat font. A site with 100 text romans, four geometric symbol fonts, three calligraphic fonts, and three dingbat fonts might need $100 \times 4 \times 3 \times 3 = 3600$ VFs.

Slots. Some glyphs have preferred slots; for example it would be useful if the letter 'A' was always in slot 65.

None of the encodings will specify bold or sans glyphs, since these are expected to be kept in separate bold or sans math fonts, with the same encoding. The glyphs which are most commonly requested in bold will be placed in the MC encoding, so if many bold glyphs are used in a document, only one extra MC-encoded family containing bold glyphs needs to be used. If few bold glyphs are requested, these can be set using macros similar to `\boldsymbol` from $\mathcal{A}\mathcal{M}\mathcal{S}$-TEX.

T_1 encoding

The T_1 (or Cork) encoding will be used for multi-letter identifiers such as 'log', 'sin' and 'lim'. Using the T_1 encoding allows arbitrary text fonts to be used for multi-letter identifiers. In many texts this will the same as the text roman, but this will not always be the case (for example Barandregt's *The Lambda Calculus*, North-Holland, 1984, has some multi-letter identifiers set in bold sans!).

This font will not normally be used for anything other than upper and lower case Latin letters. The symbol glyphs such as '+', '=' and '/' will be taken from the MS_P encoding.

Although the multi-letter identifier font will be T_1 encoded, it does not necessarily have to be a text font. In particular it may have the glyph width and italic correction adjusted to produce good subscript and superscript positioning, as long as this is not to the detriment of setting multi-letter identifiers.

Family 0 will contain a T_1 encoded font.

MC encoding

The MC encoding will contain:

- The default Latin letters (for example 'f').
- The default numerals (for example '1').
- The default punctuation (for example ',').
- The slanted and upright Greek in upper and lower case (for example 'α' and 'Γ').

Other glyphs (such as the math accents and Hebrew) will be included if there is space!

The font will also contain enough font dimensions to be used as `\fam2`, since the positioning of subscripts and superscripts depends much more on the math core font than the symbol fonts. It may also contain font dimensions for:

- Design size
- Suggested script and scriptscript design size
- Suggested values for `\mathsurround`, `\thickmuskip`, `\medmuskip` and `\thinmuskip`.

Family 2 will contain a MC encoded font.

MX encoding

The MX encoding will contain the extension glyphs from `cmex` and `ms*m`, plus frequently requested glyphs such as longer math accents, double brackets, and `\bigsqcap`.

Family 3 will contain an MX encoded font.

MS_i encodings

The MS_P, MS_1 and MS_2 encodings will contain the geometric glyphs from `cm*` and `ms*m`, plus frequently requested glyphs such as `\mapsfrom`. In addition:

- MS_P will contain calligraphic upper and lower case
- MS_1 will contain open (or 'inline' or 'outline' or 'blackboard bold') upper and lower case
- MS_2 will contain black letter (or 'fraktur') upper and lower case

There was quite a lively discussion about what to do with script upper and lower case! One possibility is to allow font implementors to replace the calligraphic letters by script letters in an MS_P font. Another is to ask that script letters be provided in T_1 encoded fonts. This point is still up for discussion.

All of the geometric glyphs used in `plain` TEX and LATEX will be kept in the MS_P encoding, so compatibility with `plain` TEX or LATEX can be achieved by loading four families, encoded as T_1, MS_P, MC and MX. Compatibility with $\mathcal{A}\mathcal{M}\mathcal{S}$-TEX or $\mathcal{A}\mathcal{M}\mathcal{S}$-LATEX can be achieved by loading six families, encoded as T_1, MS_P, MC, MX, MS_1 and MS_2.

TS_1 encoding

There are a number of text glyphs that currently live in math fonts, such as '†' or '©'. These glyphs will be put into a 'text symbol' encoding, along with the Adobe standard and expert glyphs missing from the Cork encoding, such as 'f' (florin) and '½'.

The TS_1 encoding is not designed to be used in math mode.

Work to do

At the time of writing, there is still quite a lot of work to be done!

- Propose and document the math encodings.
- Implement the math encodings with METAFONT or virtual fonts.
- Provide user interfaces for `plain` TEX and LATEX.

If you would like to help with implementing or testing the new math font standards, please write to:

`math-font-request@cogs.susx.ac.uk`

We look forward to hearing from you!

◇ Barbara Beeton
American Mathematical Society
`bnb@math.ams.org`

◇ Alan Jeffrey
University of Sussex
`alanje@cogs.susx.ac.uk`

◇ Frank Mittelbach
LATEX3 project
`mittelbach@mzdmza.zdv.uni-mainz.de`

◇ Chris Rowley
Open University
`c.a.rowley@open.ac.uk`

◇ Justin Ziegler
Ecole des Mines de Saint Etienne
`ziegler@educ.emse.fr`

An Application of Literate Programming: Creating a Format for the Bulletin of the Polish TUG

Włodek Bzyl
Uniwersytet Gdanski
Instytut Matematyki
Wita Stwosza 57
80-957 Gdansk, Poland
Internet: `matwb@halina.univ.gda.pl`

Tomasz Przechlewski
Uniwersytet Gdanski
Katedra OPD
Armii Krajowej 119/121
81-824 Sopot, Poland
Internet: `ekotp@halina.univ.gda.pl`

Abstract

This article describes the process by which the authors used WEB to create a format for the Bulletin of the Polish TUG.

Introduction

On establishing the Polish TEX Users Group the authors were appointed to create its bulletin. One of the first tasks was creation of a format for the bulletin. We wanted it to be easily maintained and fully documented. The format had to be sufficient to understand, appreciate and later to modify the code by ourselves and/or others.

It was decided that the format should be coded in WEB. Knuth's WEB System of Structured Documentation was thought to fulfill our expectations. We chose FWEB, a multi-lingual implementation of WEB by John A. Krommes (Princeton University) based on Silvio Levy's CWEB.

WEB programs are easily modified to different environments. Ideally, we should have started with an existing WEB file and then modified it via a change file. However, there were not any formats written in WEB at that moment. To that end we had to translate to WEB the format of our choice—*TUGboat* style. Oddly enough, we became pioneers.

Problem

There are only a few sets of macros for typesetting bulletins in the public domain (TTN, *TUGboat*). The *TUGboat* format is the best known, the most widely used one and can be obtained from almost all archives. *TUGboat* is designed for only one language—English. As our format is bilingual we had to modify *TUGboat* style. Apart from that, the modification was necessary because:

- CM fonts had to be replaced with Polish fonts which contain Polish diacritical characters;
- some parts of the code had to be changed, some adjusted to get a design that was slightly different from *TUGboat* design; and
- the parts of the code unnecessary for our purposes at that moment had to be removed.

Moreover, taking into account the technology of printing the bulletin we decided to use the Computer Concrete family of fonts instead of Computer Modern. It yielded decent results.

Template of WEB program

The @ symbol in the WEB language is an 'escape character'. It introduces commands. A WEB file has the form of a sequence of two elements: comments and code. They are separated by WEB commands started with @. The skeleton of a WEB file is shown below:

```
@Lx

@* Title.    ...  Some documentation

@A
TEX macros

@* Title.    ...  more documentation
```

```
@A
more TeX macros
etc...

@* Index.
```

The parts of the code introduced by the @* sequence (or @␣, @*1, @*2) are sections (unnamed sections, subsections etc.).

@Lx	switch to TeX language
@@	@ symbol
@␣	start an unnamed section
@*	start a major section
@*1, @*2	start a subsection (subsubsection)
@A	begin TeX code
@O	open new output file

Figure 1. List of frequently used WEB commands

Converting to WEB

The *TUGboat* format originally comes in three files: `tugboat.cmn`, `tugboat.sty` and `ltugboat.sty`. All files contain a lot of comments explaining the code. `tubguide.tex`, which is separately delivered, is a kind of a 'user guide' for authors. It is possible using WEB to combine all the files into one.

Unfortunately, *TUGboat* format had not been converted into WEB. So the first thing to do was the conversion. The process of making a `*.web` file is quite simple. One can do this in the following way:

- Change explanatory comments to sections by removing per cent signs and preceding the whole text with @ or @* symbols (short comments may be left untouched);
- precede macros with @A;
- double @ in macros.

The first step is the most important one as the structure of a document is decided at that moment (sections, subsections, etc.).

What is WEB

The web file is a structured document. It consists of documentation and macros simultaneously. A web file is processed with two preprocessors: TANGLE and WEAVE. TANGLE strips off documentation and reorganizes the code. WEAVE produces documentation in TeX format.

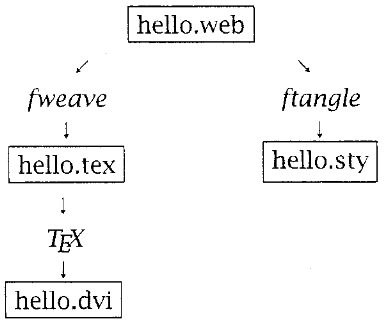


Figure 2. WEB data flow

Modifying WEB programs

Both processors, WEAVE and TANGLE, can work with two input files: *web* file and *change* file. A change file contains data which override selected portions of *web* file. The structure of a change file has the following form:

```
@x
 ... old lines ...
@y
 ... new lines ...
@z
```

Any text up to the first @x, and between @z and @x, will be bypassed (some additional comments are put there usually).

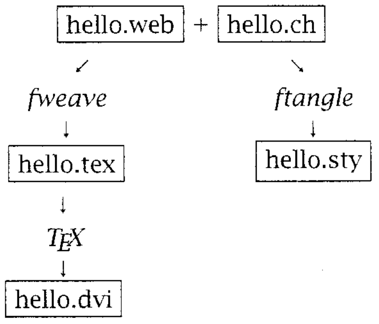


Figure 2. WEB data flow using a change file

The whole process is illustrated with the following example. Let the file `hello.web` contain the following six lines:

```
@Lx

@* First example.
@A
\def\greetings{Hello!}
@* Index.
```

and the change file `hello.ch` adapts it to the Polish language.

```
change to Polish language
@x
\def\greetings{Hello!}
@y
\def\greetings{Cze\'s\'c!}
@z
```

Conclusions

We have found this approach useful in spite of the fact that we did not use all features of WEB. Named modules are supported by FWEB, but we did not use them or the conditional exclusion/inclusion or macro definitions.

FWEB is available via anonymous `ftp` from `ftp.pppl.gov:/pub/fweb`. It runs on IBM-PC's, UNIX machines, and many other systems that provide an ANSI C compiler.

Bibliography

Knuth, Donald E. *TEX the Program.* Addison-Wesley 1988.

Knuth, Donald E. *Literate Programming.* Center for the Study of Language and Information, Leland Standard Junior University, 1992.

Krommes, John A. *The* WEB *System of Structured Software Design and Documentation for C, C++, Fortran, Ratfor, and TEX.* User Manual, 1993.

Sewell, Wayne. *Weaving a Program. Literate Programming in* WEB. Van Nostrand Reinhold, NY 1989.

Appendix

Excerpt from the tugboat.web. This will generate two files: tugboat.cmn and tugboat.sty.

```
@z======================================
%%
%% Version 1.0
%%
%% W{\l}odek Bzyl, Tomek Przechlewski
%%
%%======================================
%% original filename="tugboatc.web",
%% version="1.0",
%% date="8-July-1993",
%% filetype="TeX macros for TUGboat",
%% email="Internet:
%%     matwb@halina.univ.gda.pl,
%%     ekotp@halina.univ.gda.pl",
%% keywords="TUG, tugboat, plain tex",
%% abstract="This composite file contains
%%     the plain-based macros for preparation
%%     of TUGboat converted to WEB".
@x======================================

%% limbo

...

\def\LaMeX{{\rm L\kern-.345em
  \raise.3ex\hbox{\sc a}\kern-.16em
  M\kern-.111em\lower.6ex\hbox{E}
  \kern-.075emX}}

\def\Wtitle{GUST.WEB}

...

%--------------------------------------

@Lx   @% set the global language to \TeX

@* Identify the version.

@A
@O tugboat.cmn

\def\fileversion{v1.0}
\def\filedate{8 July 1993}

\message{File 'TUGBOAT.CMN'
    \fileversion \space\space <\filedate>}

@* Put in the index commands
   with 'at' inside.

@f '\@@ 11

@* Helpful shorthand.

@*1 Changes of category.
The following allow for easier changes
of category. These require that the character
be addressed as a control-sequence:
e.g. |\makeescape\/| will
make the |/| an escape character.

@A
\def\makeescape#1{\catcode'#1=0 }
\def\makebgroup#1{\catcode'#1=1 }

...

@O tugboat.sty

...

@* Stop reading this file if
   it's been loaded already.

@A
\ifx\tugstyloaded@@\thistubstyle
  \makeatother\initializearticle
  \endinput
\else
  \let\tugstyloaded@@\thistubstyle
\fi

\message{File 'TUGBOAT.STY'
  \fileversion \space\space <\filedate>}

@* Load macros common to \TeX\ and \LaTeX.

@A
\input tugboat.cmn

@* Some things with the same names
   as in, or reiterated from, \AmSTeX.

@A
@% override an \AmSTeX\ convention
\def\document{}

\output{\output@@}

...
```

Galleys, Space, and Automata

Jonathan Fine
203 Coldhams Lane
Cambridge
CB1 3HY
England
J.Fine@pmms.cam.ac.uk

Abstract

The thread which runs through this article is the concept of a Finite State Automaton (FSA). It is introduced as a solution to the problem of how to specify and code the amount of vertical space to be left between the various type of item which can be placed upon the page. Several methods of coding FSA are described and compared. One solution is to store the transition table and other data in the ligature table of a custom font. The best use of this method requires sofware tools which cannot readily be programmed in TEX, and also some extensions to TEX. These are discussed, and the article concludes.

Introduction

TEX forms pages by adding paragraphs and other vertical material such as skips, penalties, titles and displayed boxes to a galley, which is broken and presented to the output routine once sufficient material has been gathered. Hand setting of movable type proceeds in the same way. This article is focussed on how TEX should be instructed to insert appropriate vertical space and so forth between the paragraphs and other textual items. The proper use of space is essential to good typography.

Here are some spacing rules. Add extra space around a displayed quotation. Add extra space before a new section. Just after a section title is a bad place to break the page, so insert a penalty. Just before a subsection is a good place to break, so insert a reward — i.e., a negative penalty.

These rules do not tell us what should be done if a displayed quotation is followed by a new section. Should one use both extra spaces, or just one, or something else? It is common practice to specify 'before' and 'after' space for each element, and to take the larger of the applicable values when one element follows another. Of course, there will be exceptions. Similar considerations apply to lists. When a section title is immediately followed by a subsection, is this a good place to break the page, or a bad place? It is important to get these details right, for they can make or break the document (at the wrong place).

We shall assume that when a paragraph or other item X is added to the galley, the vertical space that should be added to the galley before X is placed upon it depends only on the sort of item that X is, together with the sort of item last placed on the galley. Thus, vertical space rules belong not to the vertical matter type itself, perhaps in *before* and *after* variants, but to combinations of vertical matter types, applying when type W is followed by type X. One can think of this as a relational approach.

If there are five types of paragraph, A to E, then there are 25 different possibilities AA, AB, AC, AD, AE, BA, BB, BC, BD, BE, CA, CB, CC, CD, CE, DA, DB, DC, DD, DE, EA, EB, EC, ED, EE, and for each of these a rule is required. Ten types of paragraph will give 100 possibilities. A large part of this article is devoted the problem of how one might specify and implement such a large collection of rules.

When TEX is typesetting the document, it needs to record the type of the last item on the galley, which we shall call the *state* of the galley. Each time an item is present for addition to the galley, we have an *event*, which may then change the state of the galley. Also, each event will cause a possibly empty *action* to take place. The action chosen will depend on the current state and on the event. In this example it is the addition of vertical space. It may be more compliciated. For example, a format may allow a section to begin on the current page if it is at most half full.

Finite State Automata

Here is a simpler example of a machine that has states, events, and actions. It is a coin-operated turnstile, such as is used by the New York subway system. One approaches the turnstile, places a coin in the slot, pushes against the bar, which then gives and allows admission. The next person will have to use another coin to gain admission. Without a coin, the bar will not move.

Here is a more formal description of the operation of the turnstile.

```
\FSA \turnstile \barred
{
  * \barred
    @ \push  \barred
    @ \coin  \open

  * \open
    @ \push  \barred     \admit_one
    @ \coin  \open
}
```

The first line tells us that `\turnstile` is a *Finite State Automaton*, whose initial state is `\barred`. (This is something we forgot to specify). The line

```
    @ \push  \barred
```

beneath `* \barred` tells us that should the `\push` occur event when the `\turnstile` is `\barred` then the state is unchanged, and the action is empty. The line

```
    @ \coin  \open
```

says that placing a coin in a `\barred` turnstile changes its state to `\open` and again has no action.

The interesting line is

```
     @ \push  \barred     \admit_one
```

beneath `* \open`, which tells us that when the turnstile is in the `\open` state (as a result of the `\coin` event), a `\push` will result in the `\admit_one` action. In specifying the operation of the turnstile, we forget to say what should happen when a coin is placed in a turnstile that is already open. In the above description, nothing happens.

The operation of the `\turnstile` has been described by the *transition lines*. These begin with an `@`, and are followed by an event, and then the new state, and then, optionally, the action for the event. To find the transition line for a given existing state X and event Y, first look for `*` followed by X in the description. This is the *label* for the state X. Now read on until you reach `@` followed by Y. This is the transition line to be followed.

The most general FSA with n states and m events will require $n \times m$ transition lines. (The acronym FSA stands for Finite State Automaton or Automata as is appropriate). In real applications this number can be reduced, by careful use of two further properties of the `\FSA` construction. The first is that the one or more labels for other events can intervene between the label for the current state, and the line for the message. In our case, the `\turnstile` will be `\open` after the `\coin` event, whatever the current state. Here is a description of `\turnstile` with only 3 transition lines.

```
\FSA \turnstile \barred
{
  * \barred
    @ \push  \barred

  * \open
    @ \push  \barred     \admit_one
    @ \coin  \open
}
```

The second property is a little more complicated. Within the rules for `\turnstile`, `#1` can be used to stand for the current state, and `#2` for the event. The FSA `\echo` described here

```
\FSA \echo \default
{
  * #1
    @ #2 #1  \message
    {
      state=\string#1,
      event=\string#2
    }
}
```

starts in the `\default` state. Whatever the event, the state is unchanged. The action is to `\message` the current state, and the event that occurred.

Here is a more sophisticated version of `\echo`. It is to have two states, `\on` and `\off`. The event `\on` is to turn change state to `\on`, the event `\off` to change state to `\off`. All other events are to leave the state unchanged, and if the state is `\on` there should be a state and event `\message` as before.

```
 1. \FSA \echo \on
 2. {
 3.   * \off
 4.     @ \on    \on
 5.     @ #2     \off
 6.
 7.   * \on
 8.     @ \off   \off
 9.     @ #2     \on   \message
10.     {
11.       state=\string#1,
12.       event=\string#2
13.     }
14. }
```

Lines 3-5 can be read as follows. If the state is `\off` and the event is `\on` then the state is changed to on, otherwise do nothing. Lines 7-10 say that if the state is `\on` and event is `\off`, then state is changed to `\off`, otherwise the state is `\on` and the `\message` is executed.

The order in which the labels and transition lines appear is very important, and may require careful thought to get the best formulation of the operation of the FSA. By way of an example, consider coding this enhancement of the `\turnstile`. It is required that the person should pass through the turnstile before some fixed period of time has elapsed since the `\coin` event. This could be requested if the turnstile is in fact a security door, and the `\coin` is the entry of an admission code.

This is an exercise, whose answer appears towards the end of the article. Please assume that there is a `\set_timer` action, which will send the `\time_out` event to the `\turnstile` at the end of the fixed time period. Please also think as to how the timer should behave if `\set_timer` is called a second time, before the `\time_out` has occurred.

A note on White Space

The macros above, and all other macros in this article, are to be read and understood in a context when all white space characters (these are space, tab, end of line, and form feed) are ignored. To allow access to space tokens, the category code of ~ is changed to 10. Knuth has programmed TEX so that such characters when read have their character code changed to that of an ordinary space (*The TEXbook*, page 47). In the code below, a ~ will produce a space character.

As usual @ will be a letter in macro files. In addition, _ will also be a letter. Math subscript, if required, can be accessed via `\sb`, which is provided in `plain` for those whose keyboards do not allow access to _.

The code fragment

```
\catcode`\ =9       \catcode`\^^I=9
\catcode`\^^M=9     \catcode`\^^L=9
\catcode`\~=10
\catcode`\@=11      \catcode`\_=11~
```

will establish this change of category codes.

Comparison with Existing Solutions

The problem of programming the vertical space between elements is scarcely discussed in *The TEXbook*.

LATEX has the very good idea of having all requests for vertical space in the document processed by special macros. This allows some resolution of conflicting rules, such as mentioned in the introduction. According to `latex.tex`, "Extra vertical space is added by the command `\addvspace{SKIP}`, which adds a vertical skip of SKIP to the document." For example, the sequence

```
\addvspace{S1} \addvspace{S2}
```

is equivalent to

```
\addvspace{maximum of S1, S2}
```

and so successive `\addvspace` commands will result in only the largest space requested being added to the page. The complicated question, as to whether `2pc` is larger or smaller than `1pc plus 2pc`, is resolved by an `\ifdim` comparison. The former is larger.

There is also a command `\addpenalty` which functions in a similar manner. The `\@startsection` command, which is the generic sectioning command for LATEX, uses these two commands to adjust the penalty and vertical space before a section, subsection, etc.

Another solution has been provided by Paul Anagostopoulos's ZzTEX. One of the most difficult tasks in creating this format, he says (1992, page 502) "was to ensure consistent vertical space between elements." His solution was to define six commands which produce vertical space. As with LATEX, all requests for vertical space should pass through these special commands.

To support these vertical spacing commands, a stack of structures is maintained. Each level of the stack records the type and amount of the previous vertical space request, penalties requested, and other data, or in other words, *the state of the galley*. Because a floating figure, for example, will add items to a galley other than the main vertical list, the state of several galleys must be recorded.

Both LATEX and ZzTEX adopt what may be called an item-based approach. Before and after a text item is added to the galley, vertical space and penalties are added or adjusted. The galley is an essentially passive object on which items are placed, and from which they are removed. This approach places restrictions on the use of the galley by the text items, for whatever is done must be capable of being undone.

In the FSA approach, responsibility is divided between the commands which form the text items, and the FSA which controls vertical space on the galley. Each text item, when it begins, passes a message to the galley. The galley then does what it will, depending particularly on the last item it processed, as recorded in its state.

For example, here is a fragment from a galley space automaton which has states `\quote`, `\text`, and `\section`.

```
* \quote
  @ \text \text \vskip 3pt~
  @ \section \section \vskip 1pc~
```

This code says that should a \quote be followed by \text, then 3 points of space are required. Should a \section follow, then 1 pica is required. Similarly

```
  * \title
   @ \quote \quote  \nobreak \vskip 3pt~
```

will inhibit a page break between a \title and a \quote, and also provide some extra vertical space.

The author's experience is that it is natural to express galley spacing rules in terms of an FSA. The \FSA construction produces code that is simple to understand, and it gathers all spacing activity into one location. When errors of implementation or deficiencies of specification arise, it is not hard to change the code to embody the improved understanding.

Those who understand SMALLTALK will realize that the galley is being modelled as an object which responds to the messages sent to it by the text item objects. The state of the galley is memory that is private to the galley, and not accessed by the text item objects except through the messages they pass to the galley. It will also be possible for the galley to send messages to or set flags for the text items, for example to suppress indentation on a paragraph which immediately follows a section title.

Performance

It is important, when writing macros that will appear frequently in the code of other programmers, or which will often be called as TEX typesets a document, that these macros be written with an eye to performance.

There are several aspects to performance. Perhaps most important is the human side. Are the macros easy to use, and do they produce code that is easy to maintain? Do they produce programs that are robust and simple to debug? Are there any traps and pitfalls for the unwary? These are the human questions.

Also a human question is speed of execution. Do the macros act quickly enough? Important here is to have an idea as to how often the macros will be called. This is not the same as how often the macros appear in the source code. Because TEX does not provide access to the current time (as opposed to the time at the start of the job), this quantity will be measured with a stopwatch.

The third measure of performance is the quantity of memory used for the storage and execution of macros. This can be crucial. Often, an enlarged version of TEX the program is required to process a document which uses both LATEX and PICTEX. The capacity of TEX is described by 14 quantities, which are listed on page 300 of *The TEXbook*.

Most important is main memory, which is used for storing boxes, glue, breakpoints, token lists, macros etc. How much space does a macro require? This question is asked and answered on page 383 of *The TEXbook*. Next most important is the hash size, which limits the number of distinct control sequence names.

TEX has a virtual memory system, by which it stores the less often used data on disc, if not enough machine memory is available. Even though TEX's memory may not be exhausted, if the total demands on the token memory exceed the actual machine memory then the resulting use of virtual memory will slow operations.

These three aspects of performance — ease of use, speed of execution, and conservation of resources — may pull in different directions. The best single goal is probably simplicity. It can bring benefit all round.

To indicate the benefits of the FSA approach, here is the \turnstile recoded using \if... to control selection of code to be executed. The state will be stored as a number by \turnstile@. Zero and one will represent \barred and \open respectively. The parameter #1 will be zero or one for \push and \open respectively. Here is one version of the code for \turnstile.

```
\def\turnstile #1
{
  \ifcase #1 ~
    \ifnum \turnstile@ = 1~
      \global \chardef \turnstile@ 0~
      \admit_one
    \fi
  \else
      \global \chardef \turnstile@ 1~
  \fi
}
```

When used, it will generate an error, because we have yet to initialise the private macro \turnstile@.

The reader may complain that although the FSA version is easier than the one just given, the use of the four new control sequences \open, \push, \barred and \coin is a cost not worth bearing. There is some merit in this. However, \open etc. are being used only as labels or delimiters, not macros. Their meaning is irrelevant, if indeed they have a meaning. Thus, existing control words could be used in their place, or the same labels shared by several FSA.

To achieve the ultimate parsimony in use of the hash table, characters can be used as delimiters. There are many character tokens available, such as x with category code 8 (subscript), which do not appear in the normal use of TeX. They can be used as delimiters. There are problems in this. The programmer would like to write \on, and have some other piece of software to consistently translate it into an unusual character token. In other areas of computing, such a program is called a preprocessor or compiler. Also required is a means of loading macros containing such unusual characters into the memory of TeX.

There is much to gain by writing TeX macros in such a manner, and the author is developing such tools. When available this problem of consumption of the hash table will disappear.

The next concern is main memory. According to the definition of the \FSA constructor which appears later, the second definition of \turnstile will produce macros equivalent to

```
\def \turnstile
  { \FSA@ \turnstile \turnstile@ \off }
```

and

```
\def \turnstile@ #1 #2
{
  * \barred
    @ \push  \barred

  * \open
    @ \push  \barred      \admit_one
    @ \coin  \open
}
```

which, according to the *The TeXbook*, page 383, will occupy 6 and 18 tokens of main memory respectively. The overhead of \FSA@ we will ignore, assuming that is shared between many FSA. By contrast, the \if... version as coded occupies 25. This could be reduced to 19 by replacing explicit numbers with the constants \z@ and \@ne. The \FSA approach seems to be superior when the specifications are more complex.

Some Other Approaches

To investigate speed of execution, an ideal problem will be coded in several ways, and then timed. The problem is that of an n-state n-event FSA, with no actions. The control sequence \state is to hold a number between 0 and $n-1$. The goal is a macro which takes a single parameter, which we shall assume is a single digit, and on the basis of this digit and the existing value of \state assign a new value to \state.

Here is the solution the author believes will be the quickest.

```
\newcount \state          \state 0

\def \quickest #1
{
  \state \csname
           \number \state #1
         \endcsname
}
```

where lines such as

```
\expandafter
\chardef \csname 00 \endcsname 3
```

define control sequences \00, \01, ... which contain the transition data. Clearly, n^2 distinct control sequences will be required to hold this table. Actions can also be supplied. With the definition

```
\expandafter
\def \csname 12 \endcsname
{
  3~
  \myaction
}
```

event 2 applied to state 1 will change the state to 3 and call \myaction.

Although quick, this approach does take a large bite out of the hash table, and so is probably not appropriate for coding the change of state as items are added to the galley. During a normal document this code will be executed perhaps 12 times each page, whereas font changes and accents will be called more often. This approach has been presented as to show how quickly TeX can do the calculation, if resources are no limitation.

There is another context in which TeX keeps a record of the state, and adjusts the action in terms of what follows. As it typesets a word, one letter after another, it consults the information that is stored in the .tfm file, to produce kerns and ligatures. TeX has been carefully programmed to store this information compactly, and to access it at high speed while processing the characters. This is the famous ligature and kerning, which has become yet more powerful with version 3 of TeX.

Suppose that in a font one wants 2 units of extra space between the characters whenever a 0 is followed by a 4. The property list .pl file below contains the script for the character 0. Line 7 says that when a 0 is followed by a 4, a kern of 2 design units should be inserted. This process is similar to the operation of the \FSA construction. It is clear that the transition table of an actionless automaton can be stored in the ligature table of such a font. More details of ligatures and kerning may be found in *The* METAFONT*book*, and Knuth (1989, 1990).

```
1. (LIGTABLE
2.    (LABEL C 0)
3.    (KRN C 0 R 1)
4.  (KRN C 1 R 3)
5.  (KRN C 2 R 1)
6.  (KRN C 3 R 4)
7.  (KRN C 4 R 2)
8.  (STOP)
9.)
```

A font needs characters as well as ligatures and kerns. Normally, the characters of a font are defined, and then the kerns and ligatures are something of an afterthought. For fonts that encode FSA, the characters are the afterthought. The lines

```
(CHARACTER C 0)
(CHARACTER C 1)
(CHARACTER C 2)
(CHARACTER C 3)
(CHARACTER C 4)
```

will supply 0, 1, 2, 3 and 4 as characters for this font, all with zero height, depth, and width. Now encode the remaining transitions into the ligature table, and call the font `fsa5`. This font encodes a 5 by 5 FSA.

The problem now is to access this data from within TEX. We shall assume that the design size and design unit of the font are both 1pt. This makes the examples easier. First load the font `fsa5`

```
\font \fsa fsa5~at~1sp
```

at one scaled point. The width of the box

```
\hbox { \fsa 04 }
```

in scaled points will be transition table entry for state 0 and event 4. As before `#1` will be a digit. The `\state` will be a `\chardef`. The macro

```
\def \quicker #1
{
  \setbox \zero \hbox
    { \fsa \number \state #1 }
  \chardef \state \wd \zero
}
```

uses the transition table to determine the new `\state`.

Finally, here is a variant of the `\FSA` construction, specially adapted to this ideal problem. The transition data is stored in the table `\slow@`, where the x's indicate where the values should be placed.

```
\def \slow@
{
  *0 @0x @1x @2x @3x @4x
  *1 @0x @1x @2x @3x @4x
  *2 @0x @1x @2x @3x @4x
  *3 @0x @1x @2x @3x @4x
  *4 @0x @1x @2x @3x @4x
}
```

The FSA itself will use the numerical value of `\state` as a delimiter. This is the reason for the `\expandafters`.

```
\def \slow #1
{
  \expandafter \def
  \expandafter \next
  \expandafter ##  \expandafter 1
  \expandafter *
               \number \state
               ##2
               @ #1
               ##3
               ##4
               ;
  { \chardef \state ##3 ~ }

  \expandafter \next \state@ ;
}
```

Finally, for those who are not in a hurry, here is the same FSA coded using the `\FSA` constructor

```
\FSA \slowest 0
{
  *0 @0x @1x @2x @3x @4x
  *1 @0x @1x @2x @3x @4x
  *2 @0x @1x @2x @3x @4x
  *3 @0x @1x @2x @3x @4x
  *4 @0x @1x @2x @3x @4x
}
```

where as before the x's indicate where the transition table should be entered.

Speed of Execution

The macros `\quickest`, `\quicker`, `\slow`, and `\slowest` will now be timed.

Even the slowest macro executes in a fraction of a second. The stopwatch will be applied not to one application of a macro, but hundreds or thousands. Rather than use a `\loop`, which will introduce considerable overheads of its own into the elapsed time, the lines

```
\let  \0 \relax
\def  \1 {\0\0\0\0\0\0\0\0\0\0}
\edef \2 {\1\1\1\1\1\1\1\1\1\1}
\edef \3 {\2\2\2\2\2\2\2\2\2\2}
\def  \4 {\3\3\3\3\3\3\3\3\3\3}
\def  \5 {\4\4\4\4\4\4\4\4\4\4}
```

will be read, resulting in macros $\backslash n$ which expand `\0` exactly 10^n times, for $n = 0, 1, 2, 3, 4$ and 5. Timing tests can now be done by setting `\0` to an appropriate value, typing `\3`, `\4` or even `\5` at the console, and starting the stopwatch.

Here is a table of results for an old MS-DOS personal computer, with a 286 processor running at 10Mhz.

A	B	C	D	E
`\let \0 \relax`	\5	4.3	43	-
`\def \0 {}`	\5	14.9	149	-
`\def \0 { \relax\relax }`	\5	22.8	228	28
`\def \0 { \quickest 0 }`	\4	16.3	1630	1430
`\def \0 { \quicker 0 }`	\4	33.2	3320	3120
`\def \0 { \quicker 4 }`	\4	33.5	3350	3150
`\def \0 { \slow 0 }`	\3	10.6	10600	10400
`\def \0 { \slowest 0 }`	\3	12.0	12000	11800
`\def \0 { \bigslow 0 }`	\3	27.1	27100	26900
`\def \0 { \bigquicker 0 }`	\4	33.6	3360	3160
`\def \0 { \bigquicker 9 }`	\4	34.3	3430	3330
`\let \0 \extract`	\4	39.7	3970	3770
`\let \0 \quickextract`	\4	13.4	1340	1140

The first column provides a meaning for the control sequence \0 which is repeatedly called by \1, \2, etc. Column **B** is the number of iterations performed, and column **C** the time in seconds taken for this number of iterations. The raw time taken for each iteration, the quotient of **C** by **B**, is in column **D**. Finally, column **E** is **D** adjusted to account for the overheads involved in the timing tests.

The first three lines of this table are there to help establish a baseline. It may be surprising that the empty macro executes at about one third of the speed of the `\relax` command. A certain amount of time will be spent in expanding \0, \1, \2, \3 and \4. This time should be discounted from the raw figures, to obtain the time actually executing the macro being timed. To produce round numbers, the base line correction has been taken as −200. This is close enough, given the accuracy on the raw data, and the purpose of the table.

According to *The* METAFONT*book* (page 317) TEX will stop reading a ligature table once it has found a "hit", and thus the entries appearing earlier in the script for a given label will be found quicker than those that are later. For the font `fsa5` as used in the test, the new state is the same digit as the event, but it is coded as a 5 by 5 table, with smaller digits first. Thus, `\quicker 0` will execute just a bit quicker than `\quicker 4`. The timing tests show that the difference in time, while significant, is small in relation to the whole.

It may again be surprising that `\slow` and `\slowest` are relatively so close to each other (although twice the *difference* is approximately the time taken by `\quicker`). If performance is an issue, it seems to be better to move to the `\quicker` or `\quickest` style rather than produce a custom FSA which works through macro expansion alone.

The previous tests relate to a 5 by 5 transition table. It should be clear that for a 10 by 10 table, the `\quickest` approach will be just as rapid as before. The `\slow` macro rewritten for 10 by 10 is `\bigslow` which runs at well under half the speed. This is because there is a quadratic element in the running time. (Encoding the data requires a replacement text with over 300 = 3 × 10 × 10 tokens, each of which must be read as the helper macro searches for its delimiters.) The `\bigquicker` macro uses the font `fsa10`, which encodes the general 10 by 10 transition table. The decrease in performance is slight, compared to the 5 by 5 `\quicker` macro. Why is this? When TEX consults a ligature table, it needs to find the location of the label for the first of the two characters. It can find this data immediately, because this location is stored as part of the information for the character. There is no quadratic element in the running time.

It should be noted that adding actions to the body of a `\FSA` will further increase the execution time, even if they are not selected, because they too constitute tokens which the helper macro has to read.

The last two lines of the table give the times for utility macros `\extract` and `\quickextract`, which will be described later.

The `\FSA` approach is probably the easiest to write code for. The `\csname ... \endcsname` approach runs very quickly, but will consume the hash size. The font ligature table approach gives code that runs quite quickly, without wasting the hash size. However, it will not be so easy to write code for this approach, not least because symbolic names will not be available. Property list files are not a preferred programming language. There will be more on this later.

Finally, when it comes to size, the font approach is a clear winner. The font `fsa5`, which encodes the general 5 by 5 transition table, occupies 47 words of font information, while the replacement text for the `\FSA` approach and also the slightly quicker variant `\slow` both consist of 85 tokens. The font `fsa10` encoding the general 10 by 10 table occupies 132 words of font information.

The next section will show how the action as well as the new state can be recorded by and recovered from the kern.

Exploring Fonts

Here are some of the important facts, concerning TEX's capacity for handling fonts. A font can contain up to 256 characters. The maximum number of fonts that may be loaded depends on the implementation, and is commonly 127 or 255.

The total amount of font information that can be stored might be 65,000 pieces. Each ligature or kern occupies one such piece. Before TEX 3 and METAFONT 2.7, there was an effective limit of 256 on the number of ligatures or kerns in a single font, but now more than 32,000 are possible. In addition, TEX 3 has smarter ligatures, which in particular allow the ligature tables for several characters to share common code.

A kern can be positive or negative, and must be smaller in magnitude than 2048pt. It can be specified with a precision of 1sp (scaled point), of which there are $65536 = 2^{16}$ in a single point. Thus there are $268435455 = 2^{28} - 1$ possible different values for a kern. Previously, the transition table of a FSA has been encoded by setting the kern for a character pair to be the new state, as a digit. Instead, the ASCII code for the new state could be stored in the kern, along with a lot more information.

In addition, each character has a height, a depth, a width, and an italic correction. Although each character in a font can have its own width, there can be at most only 15 different nonzero heights, nonzero depths, and nonzero italic corrections in a single font. Each font has at least seven dimensions, accessed in TEX through the `\fontdimen` primitive. A font can have many more such dimensions; at least 32,000 are possible.

The reader may wonder how to use the wealth of digital information in a font. Here is one method. It assumes that the kern is to have positive width, and to be a nine-digit number when writen in units of sp. This kern gives the width to `\box\zero`.

```
\wd\zero 123456789 sp  % sample value
\def \extract
{
  \expandafter
  \extract@ \number \wd \zero
}

\def \extract@ #1#2#3#4#5#6#7#8#9
   { \chardef\state #2#3#4~ }
```

Here is another. The control sequence `\count@` is a count register dedicated to scratch purposes. Note the one million is too large to be stored as a `\mathchar`.

```
\newcount \million
\million 1000000

\def \quickextract
{
  \count@ \wd \zero
  \divide \count@ \million
  \chardef \state \count@
}
```

Speed of execution for these macros is respectable, and indicates that even when the extraction time is figured in, the font approach will run quicker than the `\FSA` approach, except perhaps for the very smallest examples. For the moment it is enough to know that digital extraction is practical. What will be best will depend on the demands of the applications.

A more substantial problem is this. There may be only 127 fonts available, or perhaps up to 255 if a really large version of TEX is used. The fonts which encode FSA will never get into the final `.dvi` file (unless the macros are not working properly) and so will not trouble the `.dvi` device driver. However, to use 10 or 15 of the precious allocations of fonts for the coding of FSA may be too much. Fortunately, the same font can be used to store several FSA. In fact, as long as no FSA has more than 255 distinct events, the limit is on the total number of states, across the various FSA being packed into the font. It seems that in practice most FSA will have rather more events than states.

Implicit in this discussion is the assumption that TEX 3 is being used. Earlier versions of TEX are limited to 255 kerns and ligatures for each font. This may be enough for particular applications but even those that do may grow beyond this limit over the course of time. While compatibility with TEX 2 is desirable, it should not be required.

Is It Practical?

This article began by defining the concept of a finite state automaton. The one implementation, via the `\FSA` macro is easy to code, moderate in its use of resources, and relatively slow to run. The other, via font ligature and kerning tables, is quick to run, impressively economical in resources, and difficult to program without special tools.

The state of the galley, as we have defined it, must be recalculated with every new paragraph, heading, etc. Although not a rare event, it is not so ubiquitious that the very best performance is demanded.

Use of the font method will require special tools that translates code, perhaps written with a TEX like syntax, into a property list file from which the `.tfm` file can be produced using the program `pltotf`, together with some TEX code for handling the special actions which cannot be encoded in the font information.

Such a tool would not be tremendously difficult to write, but to ensure that it is available to all TEX users on all platforms is another matter. The only programming language a TEX user can be sure to have is TEX itself! (one can also hope that the user has BIBTEX and `makeindex` available.) This seems to force one to write the tool in TEX. But then it will be interpreted, not compiled, and so perhaps slow to run. There have been complaints about TEX as a programming language, some perhaps well founded. It is possible to write a compiler/preprocessor in TEX itself, but it is not as it now stands a tool well adapted to this task.

This line of thought, which is relevant to discussion of the future of TEX, will be investigated further in a separate article.

Solution to Exercise

Here is the solution to the `\turnstile` problem with timer. The following rules govern the behaviour. Whenever `\coin` occurs, the state becomes `\open` and action is `\set_timer`, irregardless of the existing state. Any other event, including `\time_out`, should result in the `\barred` state and no action, with one exception. If the state is `\open` then a `\push` will still `\admit_one` and result in a `\barred` state.

```
\FSA \turnstile \barred
{
  * \open
    @ \push  \barred     \admit_one

  * #1
    @ \coin  \open       \set_timer
    @ #2     \barred
}
\endverbatim
%%
```

The `\set_timer` macro requires a small amount of numerical information. It must record the number of clock ticks since the turnstile last became `\open`. The `\timer` FSA will have two states, `\off` and `\on`. It will respond to events `\on`, `\off`, and `\tick`. The `\off` event is for internal use only, to turn the `\timer` off when time has run out. Also for internal use is `\counter`, which counts the `\tick` events since the `\timer` was turned `\on`. The `\set_timer` macro in `\turnstile` should send the `\on` message to the `\timer` automaton.

```
\FSA \timer \off
{
  * \on
    @ \tick \on
        \ifnum \counter < 100~
          \global \advance \counter 1~
        \else
          \timer \off
          \turnstile \time_out
        \fi
  * \off
    @ \on    \on     \global \counter 0~
    @ #2     \off
}
\endverbatim
%%
```

Finally, it should be noted that the timer will still receive the `\tick` event from the clock, whether it is `\on` or `\off`. The last two transition line say that when `\off` the timer responds only to the `\on` event.

Coding the `\FSA` Macros

The `\FSA` macro depends on some helper macros `\FSA@` and `\FSA@@` which are similar to the `\CASE` and `\FIND` macros which the author has defined elsewhere (Fine, 1993).

The definition of `\FSA` is a little complicated. The command

```
\FSA \mymacro \initial_state { ... }
```

will first of all define the `\mymacro` to have expansion

```
\FSA@ \mymacro \FSA\mymacro
               \initial_state
```

where `\FSA\mymacro` *is a single control word.* Next, `\FSA\mymacro` is defined to be a two parameter macro (this allows #1 and #2 to appear in the transition table of the FSA) whose expansion is indeed the transition table.

These goals are accomplished by the following definition.

```
\def\FSA #1 % name of FSA
         #2 % initial state
         #3 % transitions
{
  % define the FSA
  \edef #1
  {
    \noexpand \FSA@
    \noexpand #1  % name of FSA
    \expandafter\noexpand
    \csname FSA\string #1 \endcsname
    \noexpand #2  % initial state
  }

  % define the transitions store
  \expandafter\gdef
  \csname FSA\string #1 \endcsname
      ##1 % <state>
      ##2 % <event>
  { #3 }
}
```

With these values, the result of expanding `\mymacro \event` is

```
\FSA@ \mymacro \FSA\mymacro
      \state \event
```

and so a start can be made on the coding of the helper macro `\FSA@`. It must expand the transitions store \FSA\mymacro, passing \state and \event as parameters, and then look within it for the current state and the following transition line. The scratch helper macro \next will search the expansion of \FSA\mymacro. This macro can readily find the \state and the \event, and from this the new state. To find the action is more difficult, because the action portion is not delimited by a fixed token. The next token after the action may be a @, or it may be a *. Or it may be that the transition line selected is at the very end of the transition store. If not, then the rest of the transition store must be discarded. To help take care of these possibilities, helpful delimiters are placed after \FSA\mymacro.

Here is the code for \FSA@. Like \CASE and \FIND, it is a selector macro.

```
\def\FSA@ #1 % name
          #2 % transitions store
          #3 % <state>
          #4 % <event>
{
  \def\next  ##1 % discard
           * #3 % find <state>
            ##2 % discard
           @ #4 % find <event>
            ##3 % new state
            ##4 % <action> + rubbish
             @  % next transition line
  {
    % redefine the FSA
    \gdef #1 % name
    {
      \FSA@
         #1 % name
         #2 % transitions store
        ##3 % new state
    }

    % prepare to extract the action
    \FSA@@ ##4 *  % may need this *
  }

  \expandafter
  \next #2 % transitions store
        #3 % <state>
        #4 % <event>
         @ % may need a trailing @
     \FSA@ % final delimiter
}
```

Careful thought is required to follow the execution of \FSA@. First \next is defined, with the existing state and the event as delimiters. This allows \next to extract information from the transition store. In fact, \next must find the current state, and then the event which occured, and then extract the action, and then discard the rest of the transition store, and then execute the action.

The transition store should be expanded, with the state and event as parameters, before \next is called. The part of the transition store which lies *before* the new event and action should be discarded, as should the part which lies *after* the event and action. The delimiters supplied to the definition of \next will discard the *before* portion. The new state is found immediately after the old state, in the expansion of \next, but the action is not so easy. It may be delimited by * or by @, depending on whether what follows is another line for the same state, or the script for another state. If the action is the last line of the transition store, the action will not have a delimiter at all. This range of possibilities is a consequence of the flexible syntax allowed in the \FSA command. The trailing tokens @ \FSA@ at the end of the expansion of \FSA@ are there to allow the action to be extracted, and the rest of the transition store to be discarded.

The execution of \next will result in \mymacro being defined. The new value will be the same as the old, except that \state will have been replaced by the new state. Now to extract the action, which lies in the tokens formed by the expansion of the transition store, which were not absorbed by \next. Suppose that the action had been delimited by an * rather than an @ in the transition store, or even by nothing at all. In either of these cases, the action — which is picked up and copied by \next — will now be delimited by a *. So, all that remains is to pick up the action, throw away the rubbish, and perform the action. This is done by a final helper macro.

```
\def\FSA@@  #1    % <action>
             *    % delimiter
            #2    % <discard>
         \FSA@    % delimiter
        { #1 }    % at last, the action
```

The very last token(s) in the expansion of \mymacro come out to be the action for the curent state and event, as determined by the transitions store. The action is not called until the FSA has finished *its* activities. Thus, the action can take parameters, if need be. This may be helpful. In SMALLTALK, messages are allowed to have parameters.

Bibliography

Anagnostopoulos, Paul. "ZzTEX: A macro package for books", *TUGboat*, **13** (4), pages 497 - 505, 1992.

Fine, Jonathan. "The \CASE and \FIND macros", *TUGboat*, **14** (1), pages 35 - 39, 1993.

Knuth, Donald E. "The new versions of TEX and METAFONT", *TUGboat*, **10** (3), pages 325 - 328, 1989.

Knuth, Donald E. "Virtual Fonts: More Fun for Grand Wizards", *TUGboat*, **11** (1), pages 13 - 23, 1990.

Syntactic Sugar

Kees van der Laan
Hunzeweg 57, 9893PB, Garnwerd, The Netherlands, 05941-1525
Internet: `cgl@riscl.rug.nl`

Abstract

A plea is made for being honest with TEX and not imposing alien structures upon it, other than via compatible extensions, or via (non-TEX) user interfaces to suit the publisher, the author, or the typist. This will facilitate the process of producing (complex) publications effectively, and typographically of high-quality.

Introduction

TEX is a formatter and also a programming language. TEX is unlike traditional high-level programming languages. It is still powerful, in a class of its own, unusual, and unfamiliar.

Because TEX is different, macro writers propose harnessing it into a more familiar system, by imposing syntaxes borrowed from various successful high-level programming languages. In doing so, injustice to TEX's nature might result, and users might become intimidated, because of the difficult—at least unusual—encoding used to achieve the aim. The more so when functional equivalents are already there, although perhaps hidden, and not tagged by familiar names. This is demonstrated with examples about the loop, the switch, array addressing, optional and keyword parameters, and mouth versus stomach processing.

Furthermore, TEX encodings are sometimes peculiar, different from the familiar algorithms, possibly because macro writers are captivated by the mouth processing capabilities of TEX. Users who don't care so much about TEX's programming power but who are attracted by the typesetting quality which can be obtained with TEX as formatter, can be led astray when, while searching for a particular functionality, they stumble upon unusual encodings. They might conclude that TEX is too difficult, too error-prone and more things like that and flee towards Word*whatever*, or embrace Desk Top Publishing systems.

The way out is education, next to the provision of compatible, well-documented and supported user interfaces, which don't act like syntactic sugar, by neglecting or hiding the already available functional equivalents. Neither the publication of encodings nor the provision of encodings via file servers or archives — although a nice supporting feature for the TEXies — is enough. The quality, compatibility and the simplicity of the (generic) macros should be warranted too.

It is not the aim of this paper to revitalize a programming languages notation war, but to stimulate awareness and exchange ideas.

First, I'll glance at the big collections, and after that I'll dive into the details of macros from various sources.

In the Large

Let me first look roughly at the big collections, and refer for more details to papers on the issue.

In my opinion the math mark-up in Spivak's AMS-TEX is syntactic sugar. It claims to be essentially simpler than plain's math mark-up, which it is not. It is just different and does not provide more facilities than plain. A proof? All the examples provided in "The Joy of TEX" can be formatted equally within plain. In LAMS-TEX the table part and the commutative diagrams are substantial extensions of plain, next to the general symbolic reference scheme. For more details see my book review of Spivak's Œuvre.

I consider `amsppt.sty` and the like an adaptation by a publisher of manmac for production, with value added, if not for the user's guides and the provided support. These latter things can't be overestimated for Ben Lee User[1] in my opinion. For more detail see my *AMS BLUes*.

Furthermore, I consider LATEX as syntactic sugar, especially the math part. LATEX 2.x is even more dangerous because it claims to be perfect, which it is not. If I compare the mark-up in the spirit of *The TEXbook* with my mark-up obtained via LATEX, then the latter is much more verbose and has not added much. The extras like the picture environment, symbolic and cross-referencing, and the bibliography

[1] From *The TEXbook* fame, I like the nickname BLU.

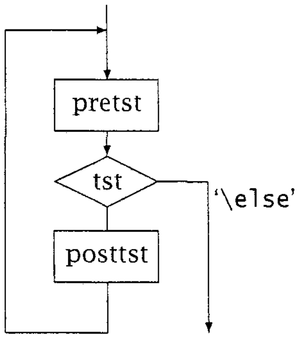
environment, can be added easily by independent tools in a manmac-like basis, when needed.[2] Multi-column issues have in general their difficulties — more likely buses-and-weirdness effects will occur; see Richard Southall's contribution about theseissues at this conference — but if one is willing to adapt proofs by hand now and then, it can be added because the functionality is available as a separate tool, nowadays.

> If only manmac and Knuth's other example formats had been appropriately documented in (additional) user's guides, then the (LA)TEX world would have looked much different from what it is today.

In the Small

In the sequel I'll descend into detail and discuss: loops, switches, array addressing, optional and keyword parameters, mouth processing, sorting and lexicographic comparison.

Loops. Knuth's loop, (*The TEXbook*, p. 219), implements the general flow



with (pseudo) syntax

`\loop`⟨*pretst*⟩`\if...`⟨*posttst*⟩`\repeat`.

Special cases result when either ⟨*pretst*⟩ or ⟨*posttst*⟩ is empty. The former is equivalent to, for example, PASCAL's **while** ... **do** ..., and the latter to **repeat**... **until**. With this awareness, I consider the variants as proposed by, for example, Pittman (1988) and Spivak (1991) as syntactic sugar.

If `\ifcase...` is used, then we have for ⟨*posttst*⟩ several parallel paths, of which one — determined dynamically — will be traversed. Provide and choose your path! What do you mean by traversing the `\else`-path?

[2] With respect to the mark-up of the list of references it is such a waste that every author should supply the full mark-up. Why not just supply references to the database of pre-formatted entries, in possession of and maintained by the editors?

Why another loop? Kabelschacht (1987) and Spivak (1989, 1991) favour a loop which allows the use of `\else`.[3] I have some objections to Kabelschacht's claim that his loop is a *generalization* of plain's loop.

First, it is not a generalization, just a clever, but variant, implementation of the loop flow chart. Second, it is not compatible with plain's loop. His exit path is via the `\then` branch (or via any of the `\or`-s, when `\ifcase` is used), and not via the `\else` branch.

The reason I can think of for introducing another loop, while the most general form has been implemented already, is the existence of commands like `\ifvoid`, and `\ifeof`, and the absence of their negatives `\ifnonvoid` and `\ifnoneof`, respectively. In those cases we like to continue the loop via the `\else` branch. For the latter case this means to continue the loop when the file is *not* ended. This can be attained via modifying the loop, of course, but I consider it simpler to use a `\newif` parameter, better known as 'boolean' or 'logical' in other programming languages. With the `\newif` parameter,[4] `\ifneof`, the loop test for an end of file—functionally ¬`\ifeof`—can be obtained via

```
\ifeof\neoffalse\else\neoftrue\fi\ifneof
```

For an example of use, see the Sort It Out subsection. Related to the above encoding of the logical ¬, are the encodings of the logical and, ∧, and or, ∨, via

Functional code	TEX encoding
`¬\if...`	`\if...\notfalse\else`
	`\nottrue\fi\ifnot`
`\if...∧\if...`	`\andtrue\if...\if...`
	`\else\andfalse`
	`\else\andfalse\fi\fi`
	`\ifand`
`\if...∨\if...`	`\ortrue`
	`\if...\else\if...\else`
	`\orfalse\fi\fi    \ifor`

with the `\newif`-s: `\ifnot`, `\ifand`, and `\ifor`.

Nesting of loops. Pittman (1988) argued that there is a need for other loop encodings.

> 'Recently, I encountered an application that required a set of nested loops and local-only assignments and definitions. TEX's `\loop...\repeat` construction proved to be inadequate because of the requirement that the inner loop be grouped.'

[3] Their loops are equivalent to the general form of the loop with the execution of an extra part after the loop.

[4] Be aware that the implementation of `\newif` does not allow for `\global`.

If we take his (multiplication) table—I like to classify these as deterministic tables, because the data as such are not typed in—to be representative, then below a variant encoding is given, which does not need Pittman's double looping. The table is typographically a trifle, but it is all about how the deterministic data are encoded. My approach is to consider it primarily as a table, which it is after all. Within the table the rows and columns are generated, via recursion, and not via the `\loop`. Furthermore, I prefer to treat rules, a frame, a header and row stubs as separate items to be added to the table proper, (van der Laan, 1992c). The *creation* of local quantities is a general TEX aspect. I too like the idea of a hidden counter, and the next best TEX solution via the local counter. The local versus global creation of counters is a matter of taste, although very convenient now and then. The creation of local quantities is tacitly discouraged by Knuth's implementation, because there is no explicit garbage collector implemented and therefore no memory savings can be gained. The only thing that remains is protection against programming mistakes, which is indeed important.

Pittman's table, focused at the essential issue of generating the elements, can be obtained via

```
$$\vbox{\halign{&\ \hfil#\hfil\strut\cr
\rows}}$$
% with
\newcount\rcnt\newcount\ccnt\newcount\tnum
\newcount\mrow\newcount\mcol \mrow2 \mcol3
\def\rows{\global\advance\rcnt1
   \global\ccnt0 \cols
   \ifnum\rcnt=\mrow\swor\fi
   \rs\rows}
\def\swor#1\rows{\fi\crcr}
\def\cols{\global\advance\ccnt1
   \tnum\rcnt \multiply\tnum\ccnt
   \the\tnum
   \ifnum\ccnt=\mcol\sloc\fi
   \cs\cols}
\def\sloc#1\cols{\fi}
\def\rs{\cr}\def\cs{&}
```

The result is

```
1 2 3
2 4 6
```

The termination of the recursion is unusual. It is similar to the mechanism used on p. 379 of *The TEXbook*, in the macro `\deleterightmost`. The latter TEXnique is elaborated in Fine (1992) and van der Laan (1992d).

The above shows how to generate in TEX deterministic tables, where the table entries in other programming languages are generally generated via nested loops. One can apply this to other deterministic math tables — trigonometric tables for example — but then we need more advanced arithmetic facilities in TEX (or inputting the data calculated by other tools), not to mention the appropriate mapping of tables which extend the page boundaries.

For a more complete encoding see my *Table Diversions* (van der Laan, 1992c). The idea is that rules and a frame be commanded via `\ruled`, respectively `\framed`. The header via an appropriate definition of `\header`, ×, the indication that we deal with a multiplication table, in `\first`, and the row stubs via definition of the row stub list. All independent and separate from the table proper part.

A better example of a nested loop is, for example, the encoding of bubble sort as given in van der Laan (1993a).

Loops and novices. Novice TEXies find Knuth's loop unusual, so they sugar it into the more familiar **while**, **repeat**, or **for** constructs, encouraged to do so by exercises as part of courseware. From the functionality viewpoint, there is no need for another loop notation.

With respect to the **for** loop, I personally like the idea of a hidden counter, see van der Laan (1992a) or Pittman (1988). The hidden counter has been used in an *additional* way to plain's loop in, for example, van der Laan (1992a), (via `\preloop` and `\postloop`), and will not be repeated here. This method is a matter of taste, which does not harm, nor hinder, because it is a compatible extension.

And for the nesting of loops we need scope braces, because of the parameter separator `\repeat`. If braces are omitted, the first `\repeat` is mistaken for the outer one, with the result that the text of the outer loop will *not* become the first `\body`. The good way is, to make the inner `\repeat` invisible at the first loop level, by enclosing the inner loop in braces.

With non-explicit nesting — for example, the inner loop is the replacement text of a macro — we still need scope braces, because otherwise the `\body` of the outer loop will be silently redefined by the body of the inner loop.

The point I would like to get across is that there is no real need for another loop encoding. Syntactic sugar? Yes!

Switches, is there a need? Apart from the `\ifcase...` construct, TEX seems to lack a multiple branching facility with symbolic names. Fine (1992) introduced therefore

```
\def\fruit#1{\switch\if#1\is
a \apple
b \banana
c \cherry
d \date        \end}
```

I have two, or rather three, remarks to the above.

First, the 'switch'-functionality is already there. Second, Fine's implementation is based upon

> 'It is clear that `\switch` must go through the alternatives one after another, reproducing the test...'

Going through the alternatives one after another is not necessary. Third, his example, borrowed from Schwarz (1987), can be solved more elegantly without using a 'switch' or nested `\if`s at all, as shown below.

The first two aspects are related. Fine's functionality can be obtained via

```
\def\fruit#1{\csname fruit#1\endcsname}
% with
\def\fruita{\apple}
\def\fruitb{\banana} %et cetera
```

With, for example, `\def\apple{{\bf apple}}`, `\fruit a` yields **apple**.

And what about the 'else' part? Thanks to `\csname`, `\relax` will return when the control sequence has not yet been defined. So, if nothing has to happen we are fine. In the other situations one could define `\def\fruitelse{...}`, and make the `else` fruits refer to it, for example, `\def\fruity{\fruitelse}`, `\def\fruitz{\fruitelse}`, etc. When the set is really uncountable we are in trouble, but I don't know of such situations. And, the five letters 'fruit' are there only to enhance uniqueness of the names.

As example Fine gives the problem, treated by Schwarz (1987), of printing vowels in bold face.[5]

The problem can be split into two parts. First, the general part of going character by character through a string, and second, to decide whether the character at hand is a vowel or not.

For the first part use, for example, `\dolist`, (*The TeXbook*, ex 11.5), or `\fifo`, (van der Laan, 1992d).

```
\def\fifo#1{\ifx\ofif#1\ofif\fi
   \process{#1}\fifo}
\def\ofif#1\fifo{\fi}
% with to be defined by the user
\def\process#1{...}
```

For the second part, combine the vowels into a string, `aeiou`, and the problem is reduced to the question $\langle char \rangle \in$ `aeiou`? Earlier, I used the latter approach when searching for a card in a bridge hand (van der Laan, 1990).[6] That was well-hidden under several piles of cards, I presume? Recently, I have used the same method for recognizing accents and control sequences in a word, (van der Laan, 1993a). Anyway, searching for a letter in a string can be based upon `\atest`, (*The TeXbook*, p. 375), or one might benefit from `\ismember`, on p. 379. I composed the following

```
\def\loc#1#2{%locate #1 in #2
\def\locate##1#1##2\end{\ifx\empty##2%
\empty\foundfalse\else\foundtrue\fi}
\locate#2.#1\end}       \newif\iffound
```

Then `\fifo Audacious\ofif`
yields **A**ud**a**c**iou**s, with

```
\def\process#1{\uppercase{\loc#1}%
{AEIOU}\iffound{\bf#1}\else#1\fi}
```

Note that en passant we also accounted for uppercase vowels. By the way, did you figure out why a period — a free symbol — was inserted between the arguments for `\locate`? It is not needed in this example.[7] Due to the period one can test for substrings: $string_1 \in string_2$? Because, $\{string_1 \in string_2\} \wedge \{string_2 \in string_1\} \Rightarrow \{string_1 = string_2\}$, it is also possibile to test for equality of strings, via `\loc`. Happily, there exists the following straightforward, and TeX-specific, way of testing for equality of strings

```
\def\eq#1#2{\def\st{#1}\def\nd{#2}
\ifx\st\nd\eqtrue\else\eqfalse\fi}
```

For lexicographic comparison, see van der Laan (1992d, 1993a) or Raichle (1992).

Knuth's switches. Knuth needed switches in his manmac macros — `\syntaxswitch`, `\xrefswitch` and the like — (*The TeXbook*, p. 424). He has implemented the functionality via nested `\if`s. My approach can be used there too, but with some care with respect to the {-token in `\next` (read: some catcode adaptations). For example:

```
\ea\def\csname sw[\endcsname{[-branch}
\ea\def\csname sw|\endcsname{bar-branche}
\def\next{[}\csname sw\next\endcsname, and
\def\next{|}\csname sw\next\endcsname
```

yields: **[-branch**, and **bar-branche**.

For manmac see *The TeXbook*, p. 412-425, and the discussion in van der Laan (1993c).

[5] A somewhat misplaced example because the actions in the branches don't differ, except for the non-vowel part.

[6] The macro there was called `\strip`.

[7] If omitted the search for 'bb' in 'ab' goes wrong: abbb vs. ab.bb, will be searched.

Array addressing. Related to the switch, or the old computed goto as it was called in FORTRAN, is array addressing. In TEX this can be done via the use of `\csname`. An array element, for example, elements identified among others in PASCAL by `a[1]` or `a[apple]`, can be denoted in TEX via the control sequences

```
\csname a1\endcsname
\csname aapple\endcsname
```

For practical purposes this accessing, or should we say 'reading', has to be augmented with macros for writing, as given in Greene (1989) and Hendrickson (1990). Writing to an array element can be done via

```
\def\a#1#2{\ea\def\csname a#1%
\endcsname{#2}}\a{1}{Contents}
```

Typesetting (reading) via `\csname a1\endcsname` yields `Contents`, after the above.

The point I would like to make is that 'array addressing' — also called table look-up by some authors — is already there, although unusual and a bit hidden. However, we are used to things like strong type-checking, aren't we? Once we can do array addressing we can encode all kind of algorithms, which make use of the array data structure. What about sorting? See the Sort It Out subsection, for a glimpse, and the in-depth treatment in van der Laan (1993a), with O($n \log n$) algorithms, and application to glossary and index sorting.

Keyword parameters. In TEX literature the functionality of keyword parameters is heavily used. Some authors impose the syntax known from command languages upon TEX: for examples see Appelt (1987) or Siebenmann (1992). In my opinion this is syntactic sugar, because of the following rhetorical question. What is essentially the difference between

```
\ref
\key W\by A. Weil
\paper Sur ...
...
\endref
```

as detailed in Siebenmann (1992) and, for example,

```
{\def\key{W}\def\by{A. Weil}
\def\paper{Sur ...}...
\typeset}
```

The typesetting is done in the cited case by `\ref...\endref`, and in the alternative case by `\typeset`. The values for the keys are the background defaults and those temporarily redefined. Note that in both cases the order of the specifications is free and that defaults (empty) are used, for not explicitly specified values.

In my bordered table macro (van der Laan, 1992c), I could have introduced keyword parameters obeying the command languages syntax. Happily, I refrained from that. I needed several parameters. A parameter for framing, with functionalities nonframed, framed, and dotframed. A parameter for ruling, with functionalities nonruled, ruled, hruled, vruled, and dotruled. And a parameter for positioning of the elements, with functionalities centered, flushed left, and flushed right. (The first element of each enumerated list of values, acting as the default value.)

Furthermore, I decided to provide the user the possibility of *optionally* specifying a caption, a header, a rowstub list, or a footer. If any of these is not explicitly specified, then the item will be absent in print too.[8] This resembles optional parameter behaviour, but has been realized by Knuth's parameter mechanism.

In following Knuth's approach, I succeeded in keeping the encoding compact, and transparent. I find it as simple, direct, and serving the purpose extremely well.[9]

Optional parameters. Among others, in LATEX, (Lamport, 1986), the mechanism of optional parameters is used. Optional parameters are a special case of keyword parameters. Knuth used optional/keyword parameters abundantly, and called them just parameters, as opposed to arguments of macros. (Think for example of his various parameters and his `\every...s`.) So it is already there, although in an unusual way.

Another example which illustrates the arbitrariness of the syntax choice with respect to optional/keyword parameters vs. Knuth's parameters is TUGboat's `\twocol` vs. LATEX's `twocolumn` style option.

Intriguing optional parameter conventions are the general and the systematic encoding of AMS-TEX's `\nofrills`, and TUGboat.sty's `\@checkoptions`.[10]

Salomon's plain Makeindex. At NTG's '92 spring meeting David Salomon reported about adapting MakeIndex to work with plain. He used optional

[8] Another difficulty was to provide a default template, which can be overridden by the user. This was solved by the same approach.

[9] Earlier, I had a similar experience (van der Laan, 1990).

[10] More about these issues in AMS BLUes (van der Laan, 1993d) and TUGboat BLUes, (van der Laan, 1993e) respectively.

parameters, with the function as given in the following table

Source	Typeset	
	document	index
`^[abc]`	abc	abc
`^[xyz]{abc}`	abc	xyzabc
`^|abc|`	abc	abc
`^|\abc|`	\abc	\abc
`^{\abc}`	replacement text of \abc	same
`^[|\abc|!xyz]{}`	nothing	\abc, xyz

and combinations thereof.

The same functionality can be obtained via Knuth's parameter mechanism. Only one parameter is needed. Let us call this the token variable \p. The idea is that the contents of \p have to be inserted before the index-entry in the index, and *not* in the text. Some symbols can be given a special meaning, like Salomon did, for example with ! (to denote a subentry).

Salomon's Source	Alternative
`^[abc]`	`^{abc}`
`^[xyz]{abc}`	`{{\p{xyz}^{abc}}}`
`^|\abc|`	`^{|\abc|}`
`^{\abc}`	`^{\abc}`
`^[|\abc|!xyz]{}`	`{\p{|\abc|!xyz}^{}}`

In the above | denotes manmac's verbatim delimiter. The macro for ^ has to be adapted accordingly. It is beyond the scope of this paper to work that out in detail.[11] The point I like to make is that the specification can be done equally well, if not simpler, via Knuth's parameter mechanism. In manmac, Knuth provides simple mark-up facilities for writing index reminders to a file, except for comments and see..., and see also... parts. The latter can be accounted for. I have touched upon these issues in *Manmac BLUes* (van der Laan, 1993c).

Mouth vs. stomach. When one starts with macro writing in TeX one can't get around awareness of TeX's digestive processing. Mouth processing is unusual. For the moment, I consider it as a special kind of built-in pre-processing, an unusual but powerful *generalization* of the elimination of 'dead branches.'[12]

Now and then encoding is published in *TUGboat*, and other sources as well, which looks difficult, and which does not seem to reflect the familiar algorithms. Sometimes, it has become difficult, because of the sought-after processing in the mouth, see for example, Jeffrey (1990) and Maus (1991).[13] The latter author agrees more or less with what is stated above '...although the macros are hard to read...'.

What puzzles me are the following questions.

Why don't authors provide the straightforward TeX encoding, not restricted to mouth processing, as well?
Why don't they make clear the *need* for mouth processing, or should I say mouth optimization?
If so, why don't they start with the straightforward encoding and explain the adaptation steps?

Faced with the above questions myself, I would answer that it is apparently too difficult to do so.[14] Furthermore, I read and worked on the math parts, the alignment parts, the macro chapter, a substantial part of the dirty tricks Appendix D and of the example formats Appendix E of *The TeXbook*, and until now found only a comment about the *capability* of TeX's mouth processing along with the macro `\deleterightmost`. I know of the argument that there is a need for it within an `\edef`, a `\write...`, and the like. I have heard that, but from an application point of view, my obvious answer is: Isn't it

[11] The preparation of an index via TeX has gotten a new dimension since my encodings of sorting within TeX. Also the writing of general index reminders to a file has been elaborated upon. For the first, see van der Laan (1993a), and for the latter, see van der Laan (1993c).

[12] Knuth might forgive me my ignorance at this point. My brows are raised when I see published code, restricted to mouth processing, which looks so verbose and unintelligible. I definitely turn my back on it when the straightforward alternative encoding is familiar, compact, elegant and generic, despite the rumour that TeX's mouth has the programming power of the Turing machine. As it is, in my opinion, that is something different from, let us say, literate programming, to indicate a broad stream of readable programs.

[13] By the way, when do we know that something is completely processed in the mouth? Is there a check on it? Or, is it just an abstract part of the TeXnigma?

[14] And what about the efficiencies? From the viewpoint of the machine and with respect to human understanding? I have not seen the common and mouth versions of an algorithm published simultaneously, let alone have them compared with respect to timing.

possible to do the things outside those constructs, equally well, and pass through the results?

> If authors don't help me out with the above, I consider the encoding as *l'art pour l'art.* Nothing wrong with that, on the contrary. The only thing against that is that it will spread a negative image about TEX encoding, certainly not under the theoretical computer scientists, but under the day-to-day BLUe-type programmers, if not the authors who just use (LA)TEX to get their work out, beautifully.

Agreed, Maus referred to *The TEXbook*, but Jeffrey could have provided a more intelligible solution, and should have refrained from burying his method under a sort of program correctness math. At the moment, it is easier to start from scratch. I experienced that already with the encoding of: the Tower of Hanoi, typesetting crosswords, generating n-copies, lexicographic comparison, and sorting. The published encodings inspired me to develop alternatives, sure, but that should not be the aim, should it? Furthermore, I wonder how many *users* have been discouraged by those 'difficult to read' codes, especially when the familiar codes are straightforward?

n-copies. I needed Maus' functionality — avant la lettre — in typesetting a fill-in form, where a number of rows had to be repeated. Of course, my editor can do it — statically — and that served the purpose. It is easy for sure, but it does not look elegant. A straightforward use of tail recursion satisfied me better, because of the simplicity, the compactness and the elegance, at the expense of a negligible efficiency loss. See the example about the bridge form in *Table Diversions* (van der Laan, 1992c).[15] The tail recursion determines the number of copies dynamically, as do the other solutions given by Knuth, for example the nice solution via the use of `\aftergroup`, (*The TEXbook*, p. 374).[16]

Sort it out. Jeffrey's problem is: given an unsorted list of (positive) integers via symbolic names, typeset the ordered list.[17] In order to concentrate on the main issues, assume that his list adheres to Knuth's list structure (*The TEXbook*, p. 378). As example consider the list[18]

```
\def\lst{\\\ia\\\ib\\\ic}
\def\ia{314}\def\ib{27}\def\ic{1}
```

The sorted numbers **1, 27, 314,** are obtained via

```
\def\\#1{\ifnum#1<\min\let\min=#1\fi}
\def\first#1{\def\lop\\##1##2\pol{%
      \let\min=##1}\ea\lop#1\pol}
\newif\ifnoe
\loop\ifx\empty\lst\noefalse\else
     \noetrue\fi
\ifnoe  \first\lst   \lst   \min,
 {\def\\##1{\ifx##1\min\else\noexpand\\%
   \noexpand##1\fi}\xdef\lst{\lst}}%
\repeat
```

The encoding implements the looping of the basic steps

- find minimum (via `\lst`, and suitable definition of the active list separator `\\`)
- typeset minimum (via `\min`)
- delete minimum from the list (again via an(other) appropriate definition of the active list separator).

For removing a typesetted element, I was inspired by `\remequivalent` (*The TEXbook*, p. 380).[19]

The above is effective for short lists, as was the case in Jeffrey's application.[20] For longer (and random) lists, techniques of order $O(n \log n)$ are more appropriate. For plain TEX encodings for the latter see van der Laan (1993a). There it has been applied to lexicographic sorting, too.

Lexicographic comparison. Eijkhout has provided macros — focused at mouth processing — for lexicographic ordering (1991). His `\ifallchars ...\are ...\before` made ample use of `\expandafter`, and is not easily accessible for somebody with say two years of TEX experience.[21]

Hackers might go into ecstasy, but application-oriented users become discouraged. For a straightforward alternative, not restricted to mouth processing, see van der Laan (1992d). The point I'd like

[15] The complexity is of order $O(n)$, instead of $O(\log n)$, which is not important, because of the small number of copies involved.

[16] Knuth in his chart macro — for typesetting font tables — uses also the straightforward approach of supplying all the lines in `\normalchart`. He could have used recursion similar to the way I did it in the multiplication table of Pittman.

[17] I have also worked on this problem, taking care of the range notation aspects (van der Laan, 1993b).

[18] Equally-well, the comma could have been used as an active list separator, which looks more natural. I decided to adhere to Knuth's notation.

[19] I was not able to apply the parameter separator technique to locate the element to be removed.

[20] Remember that sorting based on linear search has complexity $O(n^2)$.

[21] Moreover it had a flaw, as pointed out by Bernd Raichle (1992), who presented an alternative with less `\expandafters` and an intriguing use of `\csname`.

to make is that I would have welcomed the familiar solution and the transformation steps as well.

Conclusions

It is hoped that authors who can't resist the challenge to impose syntaxes from successful programming languages upon TEX also encode the desired functionality in TEX's peculiar way, and contrast this with their proposed improvements. The novice, the layman and his peers will benefit from it.

The difficulties caused by TEX's unusual encoding mechanisms can best be solved via education, and not via imposing structures from other languages. The latter will entail confusion, because of all those varieties. Furthermore, it is opposed to the Reduced Instruction Set idea, which I like. For me it is similar to the axioms-and-theorems structure in math, with a minimal number of axioms, all mutually orthogonal.

Publishing houses, user groups, and macro writers are encouraged to develop and maintain user interfaces[22] which do justice to TEX's nature, and don't increase the complexity of TEX's components. Good examples are: TUGboat's sty files, AMS-LATEX, AMS-TEX, and LAMS-TEX, and not to forget good old manmac! Macro-TEX and the LATEX3 project are promising.

File servers and archives are welcomed, but the compatibility, the simplicity and in general the quality, must be warranted too. Not to mention pleasant documentation and up-to-date-ness.

My wishful thinking is to have intelligent archives,[23] which have in store what is locally generally needed, and know about what is available elsewhere. The delivery should be transparent, and independent of whether it comes from elsewhere or was in store. For corrections and certifications I would welcome a similar approach as ACM's loose-leaf collection of algorithms

> Any third-rank engineer can make a complicated apparatus more complicated, but it takes a touch of genius to find one's way back to the basic principles, which are normally fairly simple.
>
> E.F. Schumacher, *Small is beautiful.*

I'm happy to include the following synopsis of the TUG '93 proceedings referee

> 'The point he is trying to make is that TEX macros are software and the really difficult lessons of software engineering should be used by TEX macro writers as well. Those of us who try in software engineering are not overly successful in keeping things simple and it is not surprising that little of the right way of doing software has been included in the construction of TEX macros.'

[22] And user's guides.

[23] Essentially the trickle approach, see the Earn Network Resource Guide (1993), from the fileserver.

Acknowledgements

Włodek Bzyl and Nelson Beebe are kindly acknowledged for their help in clearing up the contents and correcting my use of English, respectively.

Bibliography

Appelt, W. "Macros with Keyword Parameters". *TUGboat*, **8**(2), 182 - 184, 1987.

Appelt, W. "TEX für Fortgeschrittene — Programmiertechniken und Makropakete". Addison-Wesley, 1988.

Beebe, N.H.F. "The TUGlib Server". *MAPS*91.2, 117 - 123, 1991. (Also in TEXline 11.)

Beeton, B.N, R.F Whitney. "TUGboat Author's Guide". *TUGboat*, **10**(3), 378 - 385, 1989. (Not completely up-to-date.)

EARN Association "Guide to Network Resource Tools", 1993.

Eijkhout, V. "TEX by Topic". Addison-Wesley, 1992.

Fine, J. "Some basic control macros for TEX". *TUGboat*13(1), 75 - 83, 1992.

Greene, A.M. "TEXreation—Playing games with TEX's mind". *TUGboat*, **10**(4), 691 - 705, 1989.

Hendrickson, A. "MacroTEX". 1989.

Hendrickson, A. "Getting TEXnical: Insights into TEX Macro Writing Techniques". *TUGboat*, **11**(3), 359 - 370, 1990. (Also *MAPS*90.2.)

Jeffrey, A. "Lists in TEX's mouth". *TUGboat*, **11**(2), 237 - 244, 1990.

Jensen, K., and N. Wirth. "PASCAL user manual and report". Springer-Verlag, 1975.

Kabelschacht, A. "`\expandafter` vs. `\let` and `\def` in Conditionals and a Generalization of plain's `\loop`". *TUGboat*, **8**(2), 184 - 185, 1987.

Knuth, D.E. "*The TEXbook*". Addison-Wesley, 1984.

Laan, C.G van der. "Typesetting Bridge via TEX". *TUGboat*, **11**(2), 265 - 276, 1990.

Laan, C.G van der. "Math into BLUes". Part I: Mourning. Proceedings TUG '91, *TUGboat*, **12**(4), 485 - 501, 1991.
Part II: Sing Your Song. Proceedings EuroTEX '91, *GUTenberg Cahiers*, **10&11**, 147 - 170, 1991.

Laan, C.G van der. "Tower of Hanoi, Revisited". *TUGboat*, **13**(1), 91 - 94, 1992.

Laan, C.G van der. "FIFO & LIFO Incognito". EuroTEX '92, 225 - 234, 1992. (Also in *MAPS*92.1. An elaborated version is FIFO & LIFO Sing the BLUes.)

Laan, C.G van der. "Table Diversions". EuroTEX '92, 191 - 211, 1992. (A little adapted in *MAPS*92.2.)

Laan, C.G van der. "FIFO & LIFO Sing the BLUes". *TUGboat*, **14**(1), 54 - 60, 1993. (Also in *MAPS*92.2, 139 - 144, 1992.)

Laan, C.G van der. "Spivak's Œuvre". *MAPS*92.1, 139 - 142, 1992.

Laan, C.G van der. "Sorting in BLUe". These Proceedings. (The complete article is in *MAPS*93.1, 149 - 170. Heap sort encoding has been released in *MAPS*92.2.)

Laan, C.G van der. "Typesetting number sequences". *MAPS*93.1, 145 - 148, 1993.

Laan, C.G van der. "Manmac BLUes — Or how to typeset a book via TEX". *MAPS*93.1, 171 - 191, 1993. (To be submitted TUG '94)

Laan, C.G van der. "AMS BLUes — professionals at work". *MAPS*93.1, 192 - 212, 1993.

Laan, C.G van der. "TUGboat BLUes — how TEXies do it". *MAPS*93.2 (In progress).

Lamport, L. "LATEX, user's guide & reference manual". Addison-Wesley, 1986.

Maus, S. "An expansion power lemma". *TUGboat*, **12**(2), 277, 1991.

Pittman, J.E. "Loopy.TEX". *TUGboat***9**(3), 289 - 291, 1988.

Raichle, B. In: V. Eijkhout. "Oral TEX: Erratum". *TUGboat*, **13**(1), p. 75, 1992.

Salomon, D. "NTG's Advanced TEX course: Insights & Hindsights". *MAPS*92 Special, 1992. Revised ≈ 500p.

Schwarz, N. "Einführung in TEX". Addison-Wesley, 1987.

Siebenmann, L. "Elementary Text Processing and Parsing in TEX—The Appreciation of Tokens". *TUGboat*, **13**(1), 62 - 73, 1992.

Spivak, M.D. "AMS-TEX — The Joy of TEX". American Mathematical Society, 1986.

Spivak, M.D. "LAMS-TEX—The Synthesis". TEXplorators, 1989.

Spivak, M.D. "LAMS-TEX Wizard's manual". TEXplorators, 1991.

Youngen, R.E. "TEX-based production at AMS". *MAPS*92.2, 63 - 68, 1992.

Sorting within TEX

Kees van der Laan
Hunzeweg 57, 9893PB
Garnwerd, The Netherlands
05941 1525
Internet: `cgl@riscl.rug.nl`.

Abstract

It is shown how sorting — numbers and lexicographic — can be done completely within TEX. Lexicographic sorting allows words with ligatures and diacritical marks. As applications I selected sorting of address labels, and sorting and compressing `index.tex`, Knuth's index reminders file. It is claimed that a set can be sorted within TEX once the ordering of the set is defined and encoded in a comparison macro, in compliance with the parameter macro `\cmp`.

Introduction

This paper is an abridged version of the original "Sorting in BLUe" which has appeared in NTG's MAPS93.1. In this version I strove to get across the flavour of what can be done within TEX with respect to sorting.

The original version — not limited by space, some 20 pages — contains the in-depth treatment, the detailed explanations, the various abstractions (parameterization over the comparison operation, sorting process, lexicographic ordering, and the handling of accents and ligatures), more examples, all the macro listings, as well as a listing of my test driver. Apart from sorting the storing of the data and the final typesetting — with its parameter for separation, and for numbers the use of range notation — is dealt with in that paper. I also included an extensive list of references.

In this paper I'll show, and now and then explain, how to use the Ben Lee User level sorting macros `\sortn` — for number sorting, `\sortaw` — for sorting ASCII words, and `\sortw` — for general lexicographic sorting. At the lower level I provided in the appendices the blue collar macros `\heapsort` and `\quicksort`.

Definitions and notations. A sequence is defined as a row of numbers or words, respectively, separated by spaces. The structure `\csname`$\langle k\rangle$`\endcsname` is associated with an array with index $k = 1, 2, \ldots, n$. To denote in the documentation a value pointed by the number $\langle k\rangle$, I made use of `\val{`$\langle k\rangle$`}`, with `\def\val#1{\csname#1\endcsname}`. Macro names take suffix -n, -w, when specific for number and word data respectively. For example, `\sortn` stands for sort numbers, `\prtw` stands for print words. I have typeset the in-line results of the examples in bold face.

I have used the shorthand notation `\ea`, `\nx`, and `\ag` for `\expandafter`, `\noexpand`, and `\aftergroup`, respectively. `\k` is used as counter to loop through the values $1, 2, \ldots, n$, the index domain. `\n` contains the maximum number of sequence elements, n. `\ifcontinue` is used for controlling loops. The macro `\seq` with end separator `\qes` stores the supplied data in the array.

For typesetting the data structure I used the macros `\prtn` and `\prtw`, respectively. These are not explained here either. Loosely speaking they typeset the array, `\1...\`$\langle n\rangle$ which contains the items, as you would expect.

Some background. The reader must be aware of the differences between

- the index number, $\langle k\rangle$
- the counter variable `\k`, with the value $\langle k\rangle$ as index number
- the control sequences `\`$\langle k\rangle$, $k = 1, 2, \ldots, n$, with the items to be sorted as replacement texts.

When we have `\def\3{4} \def\4{5} \def\5{6}` then `\3` yields **4**,
`\csname\3\endcsname` yields **5**, and
`\csname\csname\3\endcsname\endcsname` yields **6**.

Similarly, when we have `\k3 \def\3{name}` `\def\name{action}` then `\the\k` yields **3**,
`\csname\the\k\endcsname` yields **name**, and
`\csname\csname\the\k\endcsname\endcsname` yields **action**.[1] To exercise shorthand notation the last can be denoted by `\val{\val{\the\k}}`.

1. Confusing, but powerful!

Another `\csname...` will execute `\action`, which can be whatever you provided as replacement text.

Sorting of Numbers

Sorting of numbers alone is not that hard to do within TeX. To design a consistent, orthogonal, well-parameterized, and nevertheless as simple as possible set of macros is the challenge, which I claim to have attained.

Example of use. After `\input heap \input sort`

```
\seq314 1 27\qes\sortn
```

yields: **1, 27, 314.**

Design choices. The backbone of my 'sorting in an array' is the data structure

`\csname`$\langle k\rangle$`\endcsname{`$\langle k^{th} elm.\rangle$`}`, $k = 1, 2, \ldots, n$,

with k the role of array index and n the number of items to be sorted.

This encoding is parameterized by `\cmp`, the comparison macro, which differs for numbers, strings, and in general when more sorting keys have to be dealt with. The result of the comparison is stored globally in the counter `\status`.

Encoding: Input. The elements are assumed to be stored in the array `\`$\langle k\rangle$, $k = 1, 2, \ldots n$. The counter `\n` must contain the value $\langle n\rangle$.

Encoding: Result. The sorted array `\1`, `\2`, …, `\`$\langle n\rangle$, with `\val1` ≤ `\val2` ≤ … ≤ `\val`$\langle n\rangle$.

Encoding: The macros.

```
\def\sortn{\let\cmp\cmpn\sort\prtn}
%
\def\cmpn#1#2{%#1, #2  ex-
pand into numbers
%Result: \status= 0, 1, 2 if
%        \val{#1} =, >, < \val{#2}.
 \ifnum#1=#2\global\status0 \else
  \ifnum#1>#2\global\status1 \else
              \global\status2 \fi\fi}
%
\def\sort{\heapsort}.
```

Encoding: Explanation. `\cmpn` has to be defined in compliance with the parameter macro `\cmp`. `\sort` must reference to one of the blue collar sorters. `\prtn` typesets the numbers. That is all.

The above shows the structure of each of the Ben Lee User sorting macros.

Sorting: `\sortn`. A (pointer) `\def\sortn{...}` is introduced which has as replacement text the setting of the parameter `\cmp`, and the invocations of the actual sorting macro and the macro for typesetting the sorted sequence.

Comparison operation: `\cmpn`. The result of the comparison is stored globally in the counter `\status`. The values 0, 1, 2 denote =, >, <, respectively.

Exchange operation: `\xch`. The values can be exchanged via

```
\def\xch#1#2{%#1, #2 counter variables
 \ea\let\ea\auxone\csname\the#1\endcsname
 \ea\let\ea\auxtwo\csname\the#2\endcsname
 \ea\global\ea\let\csname\the#2\endcsname
 \auxone
 \ea\global\ea\let\csname\the#1\endcsname
 \auxtwo}.
```

The macro for typesetting a sequence of numbers in range notation is provided in the full paper as well as in the special short paper about typesetting number sequences, which has also appeared in NTG's MAPS93.1.

Lexicographic Sorting

Given the blue collar workers `\heapsort` and `\quicksort`, respectively, we have to encode the comparison macro in compliance with the parameter macro `\cmp`. But lexicographic sorting is more complex than number sorting. We lack a general comparison operator for strings,[2] and we have to account for ligatures and diacritical marks.

In creating a comparison macro for words, flexibility must be built in with respect to the ordering of the alphabet, and the handling of ligatures and diacritical marks.

Example of use: Sorting ASCII words.
After `\input heap \input sort`

```
\seq a b aa ab bc bb aaa\qes\sortw
```

yields: **a aa aaa ab b bb bc.**

Example of use: ij-ligatures.
After `\input heap \input sort`

```
\seq{\ij}st{\ij}d {\ij} {\ij}s in tik
     t\ij\qes\sortw
```

yields: **in tik tij ij ijs ijstijd.**

Example of use: Sorting accented words.
After `\input heap \input sort`

```
\seq b\'e b\`e \'a\'a ge\"urm geur aa a
     ge{\ij}kt be ge\"\i nd gar\c con\qes
```

2. It is not part of the language, nor provided in plain or elsewhere with the generality I needed.

```
\sortw
```

yields: **a aa áá be bé bè garçon geïnd geur geürm geijkt**.

Because of the complexity and the many details involved I restricted myself in this paper to the simplified cases: one-(ASCII)letter-words, and ASCII strings, of undetermined length.

One-(ASCII)letter-words. The issue is to encode the comparison macro, in compliance with the parameter macro `\cmp`. Let us call this macro `\cmpolw`.[3] Its task is to compare one-letter words and store the result of each comparison globally in the counter `\status`. As arguments we have `\def`s with one letter as replacement text.

```
\def\cmpolw#1#2{%#1, #2 are defs
%Result: \status= 0, 1, 2 if
%        \val{#1} =, >, < \val{#2}.
 \ea\chardef\ea\cone\ea'#1{}%
 \ea\chardef\ea\ctwo\ea'#2{}%
 \global\status0 \lge\cone\ctwo}
%
\def\lge#1#2{%#1, #2 are letter values
%Result: \status= 0, 1, 2 if #1 =, >, < #2.
 \ifnum#1>#2\global\status1 \else
   \ifnum#1<#2\global\status2 \fi\fi}
```

Example of use. After the above

```
\seq z y A B a b d e m n o p z z u v c g
     q h j I i l k n t u r s f Y\qes
\let\cmp=\cmpolw\sort\prtw
```

yields: **A B I Y a b c d e f g h i j k l m n n o p q r s t u u v y z z z.**

Explanation `\cmpolw`. In order to circumvent the abundant use of `\expandafter`s, I needed a two-level approach: at the first level the letters are 'dereferenced', and the numerical value of each replacement text is provided as argument to the second level macro, `\lge`.[4]

ASCII words. The next level of complexity is to allow for strings, of undetermined length and composed of ASCII letters. Again the issue is to encode the comparison macro, in compliance with `\cmp`. Let us call the macro `\cmpaw`.[5] Its task is to compare ASCII words and to store the result of each comparison globally in the counter `\status`.

The problem is how to compare strings letter by letter. Empty strings are equal. This provides a natural initialization for the `\status` counter. As arguments we have `\def`s with words of undetermined length as replacement text.

```
\def\cmpaw#1#2{%#1, #2 are defs
%Result: \status= 0, 1, 2 if
%        \val{#1} =, >, < \val{#2}.
 {\let\nxt\nxtaw\cmpc#1#2}}
%
\def\cmpc#1#2{%#1, #2 are defs
%Result: \status= 0, 1, 2 if
%        \val{#1} =, >, < \val{#2}.
\global\status0 \continuetrue
{\loop\ifx#1\empty\continuefalse\fi
      \ifx#2\empty\continuefalse\fi
 \ifcontinue\nxt#1\nxtt \nxt#2\nxtu
     \lge\nxtt\nxtu
     \ifnum0<\status\continuefalse\fi
 \repeat}\ifnum0=\status
 \ifx#1\empty\ifx#2\empty\else
                 \global\status2 \fi
 \else\ifx#2\empty\global\status1 \fi
 \fi\fi}
%
\def\nxtaw#1#2{\def\pop##1##2\pop{\gdef
 #1{##2}\chardef#2'##1}\ea\pop#1\pop}
```

Example of use. After the above

```
\seq a b aa ab bc bb aaa\qes
\let\cmp\cmpaw\sort\prtw
```

yields: **a aa aaa ab b bb bc.**

Explanation comparison: `\cmpaw`. The macro is parameterized over the macro `\nxt`. The main part of `\cmpaw` has been encoded as `\cmpc`.[6] (That part is also used in the general case.)

We have to compare the words letter by letter. The letter comparison is done by the already available macro `\lge`. The `\lge` invocation occurs within a loop, which terminates when either of the strings has become empty. I added to stop when the words considered so far are unequal. At the end the status counter is corrected if the words considered are equal and one of the # is not empty: into 1, if #1 is not empty, and into 2, if #2 is not empty.

Explanation head and tail: `\nxt`. The parameter macro `\nxt` has the function to yield from the replacement text of its first argument the ASCII value

3. Mnemonics: compare one letter words.
4. Mnemonics: letter greater or equal. A nice application of the use of `\ea`, `\chardef`, and the conversion of a character into a number via the quote: `'`. Note that the values of the uppercase and lowercase letters differ (by 32) in ASCII.
5. Mnemonics: compare ASCII words.
6. Mnemonics: compare character.

of the first letter and deliver this value as replacement text of the second argument.[7] The actual macro \nxtaw pops up the first letter and delivers its ASCII value — a \chardef — as replacement text of the second argument.

Sorting Address Labels

Amy Hendrickson used sorting of address labels to illustrate various macro writing TEXniques. However, she used external sorting routines. Here I will do the sorting within TEX, and enrich her approach further by separating the mark-up phase from the data base query and the report generating phases.

For the imaginative toy addresses of the three composers: Schönberg, Webern, Strawinsky, I used the following definitions.

```
\def\schonberga{\def\initial{A}
 \def\sname{Arnold}\def\cname{Sch\"onberg}
 \def\street{Kaisersallee}\def\no{10}
 \def\county{}\def\pc{9716HM}
 \def\phone{050-
773984}\def\email{as@tuw.au}
 \def\city{Vienna}\def\country{AU}}
%
\def\strawinskyi{\def\initial{I}
 \def\sname{Igor}\def\cname{Strawinsky}
 \def\street{Longwood Ave}\def\no{57}
 \def\county{MA}\def\pc{02146}
 \def\phone{617-31427}
 \def\email{igor@ai.mit.edu}
 \def\city{Boston}\def\country{USA}}
%
\def\weberna{\def\initial{A}
 \def\sname{Anton}\def\cname{Webern}
 \def\street{Amstel}\def\no{143}
 \def\county{Noord-
Holland}\def\pc{9893PB}
 \def\phone{020-
225143}\def\email{aw@uva.nl}
 \def\city{Amsterdam}\def\country{NL}}
%
%the list with active list separator \as
%to be defined by the user
\def\addresslist{\as\strawinskyi
 \as\weberna\as\schonberga}
```

For the typesetting, I made use of the following simple address label format

```
\def\tsa{%The current address info is set
 \par\initials \cname \par
 \no\ \street\ \city\par
 \pc\ \county\ \country\par}
%
\def\initials{\ea\fifo\initial\ofif}
```

```
\def\fifo#1{\ifx\ofif#1\ofif\fi#1. \fifo}
\def\ofif#1\fifo{\fi}
```

Example: Selection of addresses per country. Suppose we want to select (and just \tsa them for simplicity) the inhabitants from Holland from our list. This goes as follows.

```
\def\search{NL}
\def\as#1{#1\ifx\country\search\tsa\fi}
\addresslist
```

with result

A. Webern
143 Amstel Amsterdam
9893PB Noord-Holland NL

Example: Sorting address labels.
Amy's example can be done completely within TEX, as follows.

```
%Prepare sorting
\def\as#1{\advance\k1 \ea\xdef\csname
 \the\k\endcsname{\ea\gobble\string#1}}
%
\def\gobble#1{}
%
\k0{}\addresslist%Create array
                  %to be sorted
\n\k\def\prtw{}%Suppress default \prtw
\sortw         %Sort the list
%Typeset addresses, alphabetic-
ally ordered
\k0
\loop\ifnum\k<\n\advance\k1
 \csname\csname\the\k\endcsname\endcsname
 \vskiplex\tsa
\repeat
```

with result

A. Schönberg
10 Kaisersallee Vienna
9716HM AU

I. Strawinsky
57 Longwood Ave Boston
02146 MA USA

A. Webern
143 Amstel Amsterdam
9893PB Noord-Holland NL

7. Splitting up into 'head and tail' is treated in the TEXbook, Appendix D.2, p.378, the macro \lop. There, use has been made of token variables instead of \defs.

Sorting Knuth's Index Reminders

An index reminder, as introduced by Knuth, consists of index material to be further processed for typesetting an index. In the TEXbook, p. 424, Knuth gives the syntax of an index reminder

⟨*word*⟩␣!⟨*digit*⟩␣⟨*page number*⟩.

The reminders, one per line, are written to a file because only the OTR knows the page numbers. Knuth considered this file, `index.tex`,

'... a good first approximation to an index.'

He also mentions the work of Winograd and Paxton[8] for automatic preparation of an index. Here we will provide a second approximation to an index: the index reminders are sorted and compressed. The sorting is done on the three keys

primary key: ⟨*word*⟩
secondary key: ⟨*digit*⟩, and
tertiary key: ⟨*page number*⟩.

The compressing comes down to reducing the index reminders with the same ⟨*word*⟩ ⟨*digit*⟩ part to one, with instead of one page number all the relevant page numbers in non-decreasing order.

Example: Sorting on primary, secondary and tertiary keys.

```
\def\1{z !3 1}\def\2{a !1 2}\def\3{a !1 3}
\def\4{a !1 1}\def\5{ab !1 1}\def\6{b !0 1}
\def\7{aa !1 1}\def\8{a !2 2}\def\9{aa !1 2}
\n9\k0\kk0
%
\let\cmp\cmpir\sort\let\sepw\\\null
\hfil\vtop{\hsize2cm\noindent
 after sorting\\[.5ex]\prtw}
\hfil\vtop{\hsize2.5cm\noindent
 after reduction\\[.5ex]\redrng\prtw}
\hfil\vtop{\hsize2cm\noindent
 typeset in\\index:\\[.5ex]\prtind.}\hfil
```

The above yields[9]

after sorting:	after reduction:	typeset in index:
a !1 1	a !1 1-3	a 1-3
a !1 2	a !2 2	\a 2
a !1 3	aa !1 1, 2	aa 1, 2
a !2 2	ab !1 1	ab 1
aa !1 1	b !0 1	b 1
aa !1 2	z !3 1	⟨z⟩ 1.
ab !1 1		
b !0 1		
z !3 1		

Design. Given the sorting macros we have to encode the special comparison macro in compliance with `\cmpw`: compare two 'values' specified by `\defs`. Let us call this macro `\cmpir`.[10] Each value is composed of: a word (action: word comparison), a digit (action: number comparison), and a page number (action: (page) number comparison).

Then we have to account for the reduction of 'duplicate' index entries, and finally the typesetting has to be done.

The comparison. I needed a two-level approach. The values are decomposed into their components by providing them as arguments to `\decom`.[11] The macro picks up the components
-the primary keys, the ⟨*word*⟩,
-the secondary keys, the ⟨*digit*⟩, and
-the tertiary keys, the ⟨*page number*⟩.
It compares the two primary keys, and if necessary the two secondary and the two tertiary keys successively. The word comparison is done via the already available macro `\cmpaw`.

To let this work with `\sort`, we have to `\let`-equal the `\cmp` parameter to `\cmpir`.

The comparison macro.

```
\def\cmpir#1#2{%#1, #2 defs
%Result: \status= 0, 1, 2 if
%        \val{#1} =, >, < \val{#2}
 \ea\ea\ea\decom\ea#1\ea;#2.}
%
\def\decom#1 !#2 #3;#4 !#5 #6.{%
 \def\one{#1}\def\four{#4}\cmpaw\one\four
 \ifnum0=\status%Compare second key
   \ifnum#2<#5\global\status2 \else
     \ifnum#2>#5\global\status1 \else
       %Compare third key
       \ifnum#3<#6\global\status2
       \else\ifnum#3>#6\global\status1 \fi
       \fi
     \fi
   \fi
 \fi}
```

Reducing duplicate word-digit entries. The idea is that the same index entries, except for their page

8. Later Lamport provided makeindex and Salomon a plain version of it, to name but two persons who contributed to the development. The Winograd Paxton Lisp program is also available in Pascal.
9. The unsorted input can be read from the verbatim listing.
10. Mnemonics: compare index reminders.
11. Mnemonics: decompose. In each comparison the defs are 'dereferenced', that is, their replacement texts are passed over. This is a standard TEXnique: a triad of `\eas`, and the hop-over to the second argument.

numbers, are compressed into one, thereby reducing the number of elements in the array. Instead of one page number all the relevant page numbers are supplied in non-descending order in the remaining reminder, in range notation. The macro is called \redrng[12] and is given below.

```
\def\redrng{%Reduction of \1,...,\n, with
%page numbers in range representation
 {\k1\kk0
 \ea\let\ea\record\csname\the\k\endcsname
 \ea\splitwn\record.\let\refer\word
 \let\nrs\empty\prcrng\num
 \loop\ifnum\k<\n\advance\k1
  \ea\let\ea\record\csname\the\k\endcsname
  \ea\splitwn\record.%
  \ifx\refer\word%extend \nrs with number
    \prcrng\num
  \else%write record to \kk
   \advance\kk1 \strnrs \ea\xdef
   \csname\the\kk\endcsname{\refer{} \nrs}
   \let\nrs\empty\init\num\prcrng\num
    \let\refer\word
  \fi
 \repeat\ifnum1<\n\advance\kk1 \strnrs\ea
  \xdef\csname\the\kk\endcsname{\word{}
   \nrs}\global\n\kk\fi}}
%auxiliaries
\def\splitwn#1 !#2 #3.{\def\word{#1 !#2}%
 \def\num{#3}}
%
\def\prcrng#1{\init{#1}\def\prcrng##1{%
 \ifnum##1=\lst\else\ifnum##1=\slst
  \lst\slst\advance\slst1 \else
  \strnrs\init{##1}\fi\fi}}
%
\def\strnrs{\dif\lst\advance\dif-\frst
 \edef\nrs{\ifx\nrs\empty\else\nrs\sepn\fi
  \the\frst\ifnum0<\dif
   \ifnum1=\dif\sepn\the\lst
   \else\nobreak--\nobreak\the\lst
   \fi
  \fi}}
```

Explanation: reduction of entries. The encoding is complicated because while looping over the index reminders either the reminder in total or just the page number has to be handled. The handling of the page numbers is done with modified versions of \prc, \prtfl, called respectively \prcrng and \strnrs.[13] I encoded to keep track of the numbers in the macro \nrs, in the case of duplicate word-*digit*-entries. Another approach is while typesetting the array element to process the page numbers via \prc.

Typesetting index entries. Knuth has adopted the following conventions for coding index entries.

Mark up	Typeset in copy*	In `index.tex`
`^{...}`	...	`...␣!0␣`⟨*page no*⟩
`^^{...}`	'silent'	`...␣!0␣`⟨*page no*⟩
`^|...|`	`|...|`	`...␣!1␣`⟨*page no*⟩
`^|\...|`	`|\...|`	`...␣!2␣`⟨*page no*⟩
`^|<...>|`	⟨...⟩	`...␣!3␣`⟨*page no*⟩

* |...| denotes manmac's, TUGboat's,... verbatim.

The typesetting as such can be done via the following macro.

```
\def\typind#1{%#1 a def
 \ea\splittot#1.%
 \ifcase\digit\word\or
         {\tt\word}\or
  {\tt\char92\word}\or
  $\langle\hbox{\word}\rangle$\fi{}
 \pagenrs}
%
\def\splittot#1 !#2 #3.{\def\word{#1}%
 \chardef\digit#2{}\def\pagenrs{#3}}
%
\def\prtind{{\def\\{\hfil\break}\k\kzero
 \def\sep{\let\sep\sepw}%
 \loop\ifnum\k<\n\advance\k1 \sep
  \ea\typind\csname\the\k\endcsname
 \repeat}}
```

The typesetting of the index à la TEXbook Appendix I has been dealt with in the Grandmaster chapter of the TEXbook, p.261-263.

Epilogue

No robustness was sought. The encodings have been kept as simple and flexible as possible. As a consequence no attention has been paid to safeguarding goodies like the prevention of name confusions with those already in use by an author.

Silent redefinitions do occur when not alert. Beware!

Bibliography

Laan, C.G van der. "Sorting in BLUe.". MAPS93.1, 149-169, 1993.

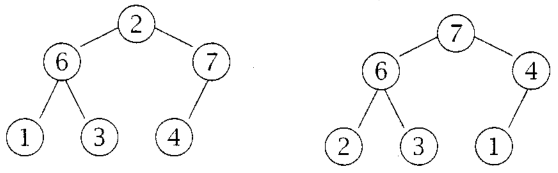
12. Mnemonics: reduce (in range notation).
13. Mnemonics: process with ranges, respectively store numbers.

Appendix: Heap Sort

The process consists of two main steps: creation of a heap, and sorting the heap. A sift operation is used in both.

In comparison with my earlier release of the code in MAPS92.2, I adapted the notation with respect to sorting in *non-decreasing* order.[14]

What is a heap? A sequence $a_1, a_2, \ldots, a_n$, is a heap if $a_k \geq a_{2k} \wedge a_k \geq a_{2k+1}, k = 1, 2, \ldots, n \div 2$, and because a_{n+1} is undefined, the notation is simplified by defining $a_k > a_{n+1}, k = 1, 2, \ldots, n$.
For example, a tree and one of its heap representations of 2, 6, 7, 1, 3, 4 read



The algorithm. In a pseudo notation the algorithm, for sorting the array a[1:n], reads

```
%heap creation
$l := n$ **div** $2 + 1$;
**while** $l \neq 1$ **do** $l := l - 1; sift(a, l, n)$ **od**
%sorting
$r := n$;
**while** $r \neq 1$ **do** $(a[1], a[r]) := (a[r], a[1])$ %*exchange*
  $r := r - 1; sift(a, 1, r)$ **od**
%sift #1 through #2
$j := \#1$
**while** $2j \geq \#2 \wedge (a[j] < a[2j] \vee a[j] < a[2j+1])$ **do**
  $mi := 2j +$ **if** $a[2j] > a[2j+1]$ **then** 0 **else** 1 **fi**
  $exchange(a[j], a[mi])$ $j := mi$ **od**
```

Encoding: Purpose. Sorting values given in an array.

Encoding: Input. The values are stored in the control sequences `\1`, ..., `\`$\langle n \rangle$. The counter `\n` must contain the value $\langle n \rangle$. The parameter for comparison, `\cmp`, must be `\let`-equal to `\cmpn`, for numerical comparison, to `\cmpw`, for word comparison, to `\cmpaw`, for word comparison obeying the ASCII ordering, or to a comparison macro of your own. (The latter macro variants, and in general the common definitions for `\heapsort`, and `\quicksort`, are supplied in the file `sort.tex`, see van der Laan (1993).)

Encoding: Output. The sorted array `\1`, `\2`, ... `\`$\langle n \rangle$, with `\val1` $\leq$ `\val2` $\leq \ldots \leq$ `\val`$\langle n \rangle$.

Encoding: Source.

```
%heapsort.tex                    Jan, 93
\newcount\n\newcount\lc\newcount\r\newcount\ic\newcount\uone
\newcount\jc\newcount\jj\newcount\jjone        \newif\ifgoon
%Non-descending sorting
\def\heapsort{%data in \1 to \n
\r\n\heap\ic1
{\loop\ifnum1<\r \xch\ic\r \advance\r-1 \sift\ic\r\repeat}}
%
\def\heap{%Transform \1..\n into heap
 \lc\n\divide\lc2{}\advance\lc1
 {\loop\ifnum1<\lc\advance\lc-1 \sift\lc\n\repeat}}
%
\def\sift#1#2{%#1, #2 counter variables
```

[14] It is true that the reverse of the comparison operation would do, but it seemed more consistent to me to adapt the notation of the heap concept with the smallest elements at the bottom.

```
 \jj#1\uone#2\advance\uone1 \goontrue
 {\loop\jc\jj \advance\jj\jj
  \ifnum\jj<\uone \jjone\jj \advance\jjone1
    \ifnum\jj<#2 \cmpval\jj\jjone
         \ifnum2=\status\jj\jjone\fi
    \fi\cmpval\jc\jj\ifnum2>\status\goonfalse\fi
  \else\goonfalse
  \fi
  \ifgoon\xch\jc\jj\repeat}}
%
\def\cmpval#1#2{%#1, #2 counter variables
%Result: \status= 0, 1, 2 if
%values pointed by
%              #1 =, >, < #2
 \ea\let\ea\aone\csname\the#1\endcsname
 \ea\let\ea\atwo\csname\the#2\endcsname
 \cmp\aone\atwo}
\endinput                    %cgl@riscl.rug.nl
```

Explanation: \heapsort. The values given in \1, . . . \⟨n⟩, are sorted in non-descending order.

Explanation: \heap. The values given in \1, ..., \⟨n⟩, are rearranged into a heap.

Explanation: \sift. The first element denoted by the first (counter) argument has disturbed the heap. Sift rearranges the part of the array denoted by its two arguments, such that the heap property holds again.

Explanation: \cmpval. The values denoted by the counter values, supplied as arguments, are compared.

Examples of use: Numbers, words. After `\input heap \input sort`

```
\def\1{314}\def\2{1}\def\3{27}\n3 \let\cmp\cmpn\heapsort
\begin{quote}\prtn,\end{quote}
%
\def\1{ab}\def\2{c}\def\3{aa}\n3  \let\cmp\cmpaw\heapsort
\begin{quote}\prtw,\end{quote}
and
\def\1{j\ij}\def\2{ge\"urm}\def\3{gar\c con}\def\4{\'el\`eve}\n4
\let\cmp\cmpw {\accdef\heapsort}
\begin{quote}\prtw\end{quote}
```

yields[15]

> 1, 27, 314,
>
> aa ab c,

and

> élève garçon geürm jij.

[15] `\accdef` suitably redefines the accents within this scope.

Appendix: Quick Sort

The quick sort algorithm has been discussed in many places. Here the following code due to Bentley has been transliterated.[16]

```
procedure QSort(L,U)
if L<U then Swap(X[l], X[RandInt(L,U)]) T:=X[L] M:=L
   for I:=L+1 to U do if X[I]<T M:=M+1 Swap(X[M], X[I]) fi od
   Swap(X[L], X[M])
   QSort(L, M-1) QSort(M+1, U)
fi
```

Encoding: Purpose. Sorting of the values given in the array \⟨*low*⟩, ..., \⟨*up*⟩.

Encoding: Input. The values are stored in \⟨*low*⟩, ..., \⟨*up*⟩, with $1 \le low \le up \le n$. The parameter for comparison, \cmp, must be \let-equal to \cmpn, for number comparison, to \cmpw, for word comparison, to \cmpaw, for word comparison obeying the ASCII ordering, or to a comparison macro of your own. (The latter macros, and in general the common definitions for \heapsort, and \quicksort, are supplied in the file sort.tex, see van der Laan (1993).)

Encoding: Output. The sorted array \⟨*low*⟩, ... \⟨*up*⟩, with \val⟨*low*⟩ ≤ ... ≤ \val⟨*up*⟩.

Encoding: Source.

```
%quick.tex                        Jan 93
\newcount\low\newcount\up\newcount\m
\def\quicksort{%Values given in \low,...,\up are sorted, non-descending.
%Parameters: \cmp, comparison.
 \ifnum\low<\up\else\brk\fi
%\refval, a reference value selected at random.
 \m\up\advance\m-\low%Size-1 of array part
 \ifnum10<\m\rnd\multiply\m\rndval
   \divide\m99 \advance\m\low \xch\low\m
 \fi
 \ea\let\ea\refval\csname\the\low\endcsname
 \m\low\k\low\let\refvalcop\refval
 {\loop\ifnum\k<\up\advance\k1
   \ea\let\ea\oneqs\csname\the\k\endcsname
   \cmp\refval\oneqs\ifnum1=\status\global\advance\m1 \xch\m\k\fi
   \let\refval\refvalcop
  \repeat}\xch\low\m
 {\up\m\advance\up-1 \quicksort}{\low\m\advance\low1 \quicksort}\krb}
%
\def\brk#1\krb{\fi}\def\krb{\relax}
\endinput                      %cgl@riscl.rug.nl
```

Explanation. At each level the array is partitioned into two parts. After partitioning the left part contains values less than the reference value and the right part contains values greater than or equal to the reference value. Each part is again partitioned via a recursive call of the macro. The array is sorted when all parts are partitioned.

In the TEX encoding the reference value as estimate for the mean value is determined via a random selection of one of the elements. The random number is mapped into the range [*low* : *up*], via the linear transformation \low + (\up − \low) ∗ \rndval/99.[17]

The termination of the recursion is encoded in a TEX peculiar way. First, I encoded the infinite loop. Then I inserted the condition for termination with the \fi on the same line, and not enclosing the main part of the macro. On termination the invocation \brk gobbles up all the tokens at that level up to its separator \krb, and inserts its replacement text — a new \fi — to compensate for the gobbled \fi.

[16] L, U have been changed in the TEX code into low, up.

[17] Note that the number is guaranteed within the range.

Examples: Numbers, words. After `\input quick \input sort`

```
\def\1{314}\def\2{1}\def\3{27}\n3 \low1\up\n\let\cmp\cmpn
\quicksort
\begin{quote}\prtn,\end{quote}
%
\def\1{ab}\def\2{c}\def\3{aa}\def\4{\ij}\def\5{ik}\def\6{z}\def\7{a}\n7
\low1\up\n\let\cmp\cmpw \quicksort
\begin{quote}\prtw,\end{quote}
and
\def\1{j\ij}\def\2{ge\"urm}\def\3{gar\c con}\def\4{\'el\`eve}\n4
\low1\up\n\let\cmp\cmpw  {\accdef\quicksort}
\begin{quote}\prtw.\end{quote}
```

yields[18]

> 1, 27, 314,
>
> a aa ab c ik ij z,

and

> élève garçon geürm jij.

[18] `\accdef` suitably redefines the accents within this scope.

Teaching Digital Typography — the Didot Project

Mary Dyson
Department of Typography & Graphic Communication
University of Reading
2 Earley Gate
Whiteknights, Reading
RG6 2AU UK
Internet: `ltsdyson@rdg.ac.uk`

Abstract

This paper briefly outlines the Didot project on teaching digital typography. One of the key issues, namely identifying the users of digital typography, is explored and related to the type of material that could be included in a curriculum for digital typography. Teaching methods and material that have been developed in this area are outlined, with particular reference to the work at Reading.

The Didot Project

Introduction. The acronym Didot stands for "DIgitising and Designing of Type" and this project has been funded as part of the European COMETT II programme. The project started in 1990 and is due to finish in September 1993. The partners come from research centres, academic institutions, commercial organisations and studios in France, Switzerland, Germany, UK, Spain, Greece and Italy.

Aims. The aims of the project are to:

- design, implement and evaluate a curriculum for digital typography, designed for both computer-oriented specialists and graphic artists and typographers;
- organise seminars and workshops for both groups; and
- publish and distribute information.

Key Issues

Promoting discussion. One of the main aspects of the project has been the provision of mechanisms for encouraging discussion between computer specialists and design specialists. Seminars and workshops have been set up to combine the teaching of both groups, but there has been a predominance of either one or the other group at the meetings.

Defining digital typography. The Didot project focuses on type design, looking at methods for creating and drawing characters. However, some of the work at Reading has extended the scope of the project to include how we use typefaces in designing documents. The study of digital typography is therefore relevant to, not only those involved in creating and manipulating fonts, but also users of document preparation systems.

Users of digital typography

In wishing to address all users of digital typography, we therefore need to consider:

- computer scientists and technicians,
- graphic designers and typographers,
- type designers,
- teachers and students, and also
- amateur designers.

This last group may not fit into either of the two categories of computer or design specialist, but do make up a large percentage of users.

Graphic artists and typographers. Having established the range of users, we should consider what material is relevant to each of the groups engaged in digital typography. Graphic artists and typographers use the tools of digital typography, and may have something to say about the development of such tools, but they are not normally the developers, as they lack the technical skills. The contribution that designers can make to developing software was discussed at a summer school in Lausanne. The nature of the tools that are being produced should be influenced by the working procedures adopted by designers. For example, designers may have clear ideas as to what type of work they wish to do on paper, and what can best be carried out on screen. However, such a contribution does not necessarily require that designers be taught the technical aspects of digital typography.

Computer specialists. Computer specialists may be the developers of the tools of digital typography, and also users. However, they may not have the appropriate design knowledge to make best use of the tools. We therefore need to identify those aspects of design that are most relevant to their aims. We may need to distinguish between what is taught to users and what is taught to developers and implementors (cf. Brown and Utting, 1992).

Amateur designers. The large group of users, classified as amateur designers, are likely to come from a range of disciplines, but have probably become users of digital typography through an interest in computers, rather than design. They may therefore be similar to computer specialists, but may have slightly different objectives in learning about digital typography.

Approaches to integration

Working together. At a Reading Didot seminar, we discussed some of the ways in which computer scientists and designers can work together. Over the course of Didot seminars, designers have received information about digital techniques, and computer specialists have heard about design issues. One of the more obvious problems is interpreting the 'language' of the other discipline. For example, design concepts are not expressed in the 'normal language' of computer scientists. We therefore need to question how far we should go in teaching another discipline. It may be most appropriate for computer scientists to use the tools (with direction from designers) as opposed to designers using the technology themselves.

Curriculum development. One way to approach this diversity of needs is to design a curriculum for digital typography which is sufficiently flexible to cover a wide range of requirements. The nature of a generic curriculum is discussed by Dyson (1992), who argues that the same topics may be relevant to different disciplines, but the subject matter may need to be treated differently depending on the background of the students.

The subjects that could be included in a curriculum for digital typography have been explored by André and Hersch (1992) who concentrate on the computer science aspect of the subject. They put forward the argument, which is fundamental to the Didot project, that digital typography should not be taught without teaching classical typography. Within the Didot project, this is dealt with in terms of historical and cultural aspects of letterforms and the fundamentals of letterforms and the design of type.

Teaching Methods and Material

The Didot project has explored a range of teaching methods and materials in relation to digital typography which include:

- seminars,
- workshops,
- short courses,
- vacation courses, and
- tools.

This paper briefly describes the full programme of seminars conducted by Didot partners so far, but focuses on examples of activities and teaching material from Reading. These examples are considered in terms of their relevance for the various groups of users of digital typography outlined above.

Programme of seminars. The nature of seminars has varied, depending on the specialisms of the seminar organisers, and also the country where they have taken place. The seminar/workshop in Reading explored ways of introducing people to some of the issues surrounding letterform design and studied digital techniques alongside traditional methods of design and manufacture. The summer school in Lausanne provided a more thorough grounding in technical matters, combined with the cultural, historical and aesthetic aspects of the subject. The Basel seminar built upon the previous seminars and developed and evaluated educational concepts. The French seminars were aimed at graphic designers and provided a means of demonstrating and working with the new technologies. The workshops of the Didot works seminar in Hamburg again focused on digital tools, with the lectures providing a rich design context.

The seminars in Italy and Greece were somewhat different in nature as they highlighted the important role of education within their respective countries. In particular, they raised awareness of the problems of using the tools of digital typography without the necessary background knowledge.

Reading seminar. The seminar in Reading with the title 'Type design: tradition and innovation' involved exercises in lettering, lectures on historical aspects of type design, and demonstrations of making type by hand (punchcutting, matrix making and type casting) and by machine (IkarusM). Those who attended were mainly involved in type design, type production or education, but there were a few computer specialists. A seminar/workshop of this nature is probably most useful for those who have a little knowledge of type design and wish to specialise. The computer-oriented participants felt that there was insufficient innovation (i.e., technology) in the programme. This comment may have reflected their own personal views, or may have been a suggestion as to what would have been more appropriate for the design-oriented group.

Local workshops. In addition to international seminars, a series of local workshops in Reading have introduced typography to beginners through the three areas of lettering, traditional handsetting and computers. The main objective of the workshops was to explore the relationship between major typographical variables through practical experience of different techniques and tools. Lettering introduces students to the influence of the tool on letterforms. Handsetting allows students to directly manipulate type and space, an experience which can then be translated into the less tangible medium of computer typesetting.

Basic issues of legibility, dealt with in theory classes, were re- examined. The relationships between choice of typeface, type size, interlinear spacing, line length, setting, hyphenation and format were explored in a series of exercises using the computer to set type. The students then evaluated the results of their exercises through conducting empirical tests.

These activities were aimed at establishing effective design procedures for digital typography, and are therefore suitable for all groups of users except experienced designers. They would be particularly appropriate for amateur designers.

Short courses. Courses spanning two to three days are regularly conducted at Reading on the subject of design issues for desktop publishing. These cover aspects of document design such as legibility, house style, heading and cueing devices, and are targeted at amateur designers.

Some of those who attend are computer-oriented and this has sometimes led to a greater interest in the tools themselves, rather than how to design using the tools. This natural tendency to seek out the areas with which we are most familiar, is one of the problems we need to overcome in Didot's interdisciplinary teaching. It may also be a warning that we should not go too far in trying to cross disciplines.

Vacation courses. Some of the teaching methods used at Reading in relation to the teaching of historical and cultural aspects of letterforms have been evaluated as part of the Didot project. Students attend two vacation courses as part of the four year BA(Hons) in Typography & Graphic Communication. One of these takes place in Northern Europe and the other in Italy. These courses abroad provide direct experience of the material they are learning about through lectures and seminars in the Department and aim to stimulate interest in the subjects they are studying. The evaluations have looked in particular at working methods, and how useful the students perceive these methods to be. Comparisons have also been made with other forms of teaching, such as lectures, seminars and practical work. The questions have evaluated:

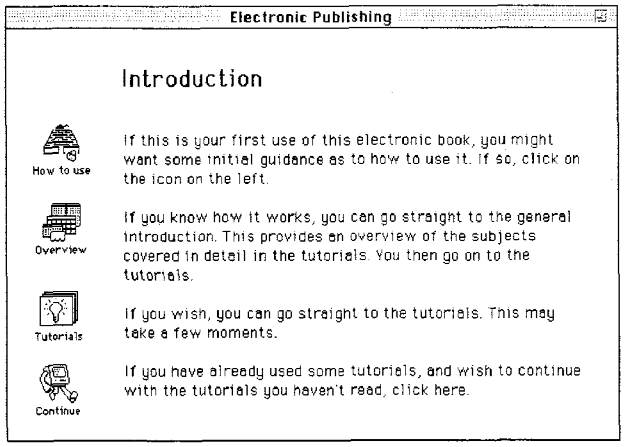
- the type of activities engaged in on the vacation courses;
- forms of preparation;
- sources of information;
- methods of learning;
- methods of recording information; and
- methods of analysis and synthesis.

The results support the use of first hand experience as a means of learning. The courses help with student's understanding of specific issues in theory and history, as well as providing inspiration for practical work. There does however, need to be sufficient preparation before the course and a means of consolidating what has been taught afterwards, to make best use of the time spent in observation and analysis whilst on the course. Although such courses would be excellent means of stimulating interest in all groups of users, they are only practical as part of a more extensive programme of teaching.

Tools. Within the context of the Didot project, specific tools have been developed as teaching material. An interactive program comprised of exercises in character-hinting techniques has been developed at EPFL.

As a student project at Reading, a video has been made based on material from the seminar. The video explains the process of punchcutting and type manufacture to people with no knowledge of the subject.

Also at Reading, a hypertext on the subject of document preparation systems has been written to support a series of lectures and is currently being evaluated. One of the introductory screens is shown in Figure 1. This tool is useful for design students as a source of reference material on some of the technological aspects of the subject (see Figure 2). However, it could be modified to suit different types of students, or the requirements of different courses. If placed within an interdisciplinary context, computer-oriented students may desire greater technical detail than typography students, but they may also require more detailed explanation



Electronic Publishing

Introduction

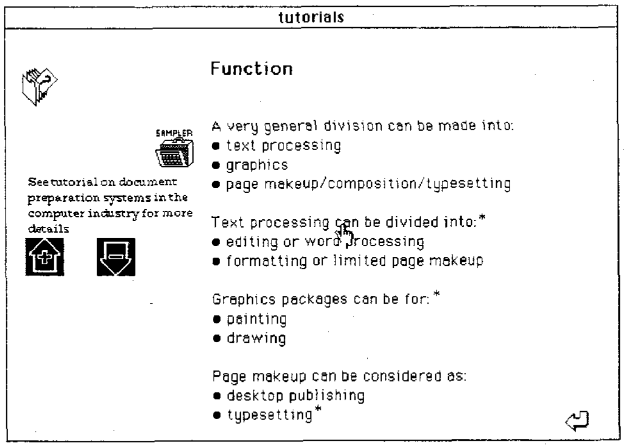
If this is your first use of this electronic book, you might want some initial guidance as to how to use it. If so, click on the icon on the left.

If you know how it works, you can go straight to the general introduction. This provides an overview of the subjects covered in detail in the tutorials. You then go on to the tutorials.

If you wish, you can go straight to the tutorials. This may take a few moments.

If you have already used some tutorials, and wish to continue with the tutorials you haven't read, click here.

Figure 1: An introductory screen of HyperCard on the subject of document preparation systems.



tutorials

Function

See tutorial on document preparation systems in the computer industry for more details

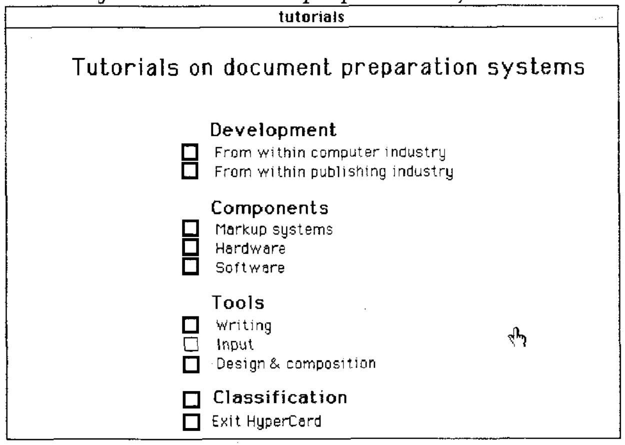
A very general division can be made into:
- text processing
- graphics
- page makeup/composition/typesetting

Text processing can be divided into:*
- editing or word processing
- formatting or limited page makeup

Graphics packages can be for: *
- painting
- drawing

Page makeup can be considered as:
- desktop publishing
- typesetting*

Figure 3: A screen from a tutorial showing that further information is available on text processing.



tutorials

Tutorials on document preparation systems

Development
- ☐ From within computer industry
- ☐ From within publishing industry

Components
- ☐ Markup systems
- ☐ Hardware
- ☐ Software

Tools
- ☐ Writing
- ☐ Input
- ☐ Design & composition

- ☐ Classification
- ☐ Exit HyperCard

Figure 2: Contents page showing the tutorials available.

of design issues. This additional information can be included at a different level, so that users can call up the information if they wish (Figure 3). Developing tools such as these may meet the needs of a range of users of digital typography.

Conclusions

Digital typography encompasses a diverse range of specialisms and we must consider the balance between teaching the core of the subject and developing and distributing specialist material. The requirements of specific types of users need to be clearly defined and mapped onto a range of appropriate teaching methods and materials.

References

André, J. and Hersch, R. "Teaching digital typography", EP-ODD, **5(2)**, 79 - 89, 1992.

Brown, H. and Utting, I.A. "Teaching EP to computer scientists", EP-ODD, **5(2)**, 91 - 96, 1992.

Dyson, M.C. "The curriculum as a hypertext", EP-ODD, **5(2)**, 63 - 72, 1992.

Russian-Speaking User: From Chi-Writer and Ventura Publisher to TEX; Learning Difficulties

Irina V. Gorbunova
International Laser Center of Moscow State University
Vorob'evy Gory, 119899 Moscow, Russia
Internet: ivg@compnet.msu.su

Why TEX is the Best

Let us start with a brief history of desktop publishing in Russia. Chi-Writer (CW) was the first russified text processor to enable an author to accomplish a composition or to prepare a camera-ready copy of a mathematical text. Based on a WYSIWYG principle, this program allows the author to work in the manner he or she was used to: that is, the result of composition is seen on the screen, which is why such work is similar to writing on paper. Thus many authors started working with CW.

The problems appear when the author is going to get a printout of the work. Laser printers, and cartridges for them, are very expensive in this country: moreover, not many organizations and authors have laser printers. To get a printout, the author brings the article to a publisher or to any other place where there is a laser printer, and then sees that the result is not whart was expected. The reasons are as follows: (1) differences in fonts used and those which the publisher possesses; and (2) the same fonts being tied to different keys. (In CW, each font is tied to the functional key on the keyboard and has its number in the fonts list. This number coincides to the number of the functional key. For example, font number one is tied to the F1 key. The number of the font can be easily changed by users; in this case the key to which the font is tied also changes. Bearing in mind the above, it is clear that there is no absolute coincidence in the programs modified by users. As a result, in the print out, the Cyrillic font may be replaced by the Latin font or the Symbol font may be replaced by the Greek. It is not very difficult to restore the order, but you have to know how to do it, and have experience, time and desire.)

Another problem, caused by the graphical nature of CW, arises when text which is prepared by CW is used in another program. This offers great difficulties for those who have no special converters.

I am not going to present in detail all the difficulties coming from using CW or explain the difference between CW and TEX, but I think that even the problems described above show that *TEX is better.*

Ventura Publisher was the second russified program to allow an author to compose a mathematical text. Ventura Publisher works on the WYSIWYG principle, but it has become an intermediary stage on the road to TEX, because of the convenience of first preparing an input file (ASCII or in a format of any program compatible with Ventura Publisher) and then importing it into the program. This way, working with Ventura Publisher, Russian-speaking users have first had to get used to working with an input file prepared by a word processor.

Ventura Publisher itself, like CW, has a large number of faults from the point of view of polygraphical quality, and in many cases the number of defects is increased if the keyboarder does not understand mathematical typography conventions. I do not want to say that in order to work with TEX it is not necessary to know the rules of composition of a mathematical text but in many cases TEX helps you, knowing the rules, and doing many things automatically.

Bearing in mind all the above we can propose once again that *TEX is much better* than CW and Ventura Publisher.

Up until now I have been reviewing the problems that authors come across working with a mathematical text in Russian. However, there are a large number of the authors who write articles or send translations of their articles printed in Russian to foreign publishers. Many of them are asked to prepare their articles in TEX and to send a `.tex` file to the editorial board. Such a requirement forces the scientists to study TEX and to work with it. In addition, it is very convenient to send a `.tex` file to a foreign publisher by e-mail, which is very important for Russian authors, due to the fact that our post does not work properly. Many of them have not worked with any desktop publishing system before and are not used to *any* system. For me, TEX was the first program I started working with and, consequently, I have had no inconvenience based on the custom of working in a WYSIWYG regime. Thus, those who are used to working with TEX and then start using any other program can not only confirm that *TEX is much better* but can judge that **TEX is the best!**

Difficulties for Russian-Speaking Users Learning TEX

One of the main difficulties for Russian-speaking users in learning TEX is the small number of books on TEX (and TEX-related software) translated into Russian. In addition, Russian-speaking users are short of literature on TEX in English. Sometimes it is impossible to get the book needed just for a few days. To buy the book you need in dollars is impossible because of the inadmissibly high rate of the dollar to the rouble (a price of a book converted into roubles is greater than a month's salary).

Another difficulty, and I think it can also be considered as one of the main ones, is the absence of TEX support. To start working with TEX you need to have an opportunity of solving the problems that appear with somebody skilled in TEX. To build a support activity properly, a support person should know the level of users' education, their experience, and abilities (Hoover, 1992).

I have spoken with many beginners and heard that the problems appear at the moment of installation of TEX. Sometimes it is impossible to install the whole package because of the lack of room on a disc. For those who do not know the structure of the package there is a risk of skipping an important directory. That is why, from the very moment of installation of TEX, many questions arise and the user should have an advisor who is able to answer questions or point to appropriate documentation. In the near future such support can be partially accomplished due to the fact that electronic mail in Russia is becoming more widespread.

Approximately a year ago, electronic mail was something very rare and very expensive, especially for international communication. During the past year some changes have taken place in this field. Several non-commercial nets are being used now, and several organizations (universities, for example) have sponsors to cover their e-mail expenses. That is why it is possible now not only to use e-mail as often as you need, but to subscribe to teleconferences, to get information and the necessary materials, and to get the answers for your questions. The members of CyrTUG are discussing the possibility of having a common electronic archive. On solving this problem it will become possible to share information from abroad between the members of the group. It is very important to have such an archive, paying attention to the fact that transmitting the information inside Russia, even via the commercial nets, is not very expensive.

In conclusion, I would say that in spite of the difficulties, TEX is spreading in Russia more and more widely (the latest CyrTUG conference confirmed this fact) and that the number of authors, scientists, and staff members of publication departments interested in TEX increases.

Bibliography

Anita Z. Hoover, "The key to successful support: knowing your TEX and LATEX users", *Proceedings of the 7th EUROTEX Conference*, Prague, pp. 71-85, 1992.

How to Set Up and Maintain a TEX System

R. Allan Reese
Computer Centre
Hull University
Hull HU6 7RX
UK
Phone: +44 482 465296
Janet: `r.a.reese@uk.ac.hull.ucc`

Abstract

TEX use is increasing, but TEX suffers from a folklore perception of being hard to use and aimed at computer programmers. This is partly because of the packaging and presentation. TEX documentation generally assumes TEX has already been set up — someone else has written a *Local Guide*. Compared with commercial software that is 'load and go', TEX installation seems to expect an unreasonable level of systems knowledge. This is compounded by the lack of a definition of what would constitute a TEX installation; it is far more than just TEX authenticated by passing the TRIP test. The continuing development and improvement of TEX-related tools distributed via networks aggravates the situation. The TEX community must specify the components and the 'current versions' that should be available in any 'standard' installation if TEX is to be used as a *lingua franca* for document transfer. The combination of TUG and CTAN organizations form a reasonable basis for carrying out this function. The paper includes a list of components that this author has assembled over several years; the next stage would be to expand this list and make the list generally available.

The problem

There are several papers and distributed notes on the general theme of 'Getting started with TEX'. Daneliuk (1993) is a list of TEX-related tutorials. Childs et al. (1988) or Martin (1990) describe introductory training in the TEX mark-up language. Doob (1992) is a `plain` TEX primer, and Warbrick (1992) and Maltby (1992) are primers to LATEX. Rahtz (1992a) is a guide to using (LA)TEX under Unix and discusses related software to put TEX in context. However, all new TEX users are referred to their *Local Guide* for the mechanics of how to use a system that is *assumed to be available*; there is little comparable help for the first person who fetches TEX to a site.

Rahtz (1992b) provides very brief notes for Unix systems, but assumes a lot of systems knowledge and ability on the part of the user if the `make` files do not work as intended. Luecking (1993) summarizes on email the process and problems of setting up the widely used IBM PC version emTEX, but does not cover what should constitute a 'TEX system'. He writes, "I have assumed that the reader is familiar with the general idea of a TEX system [and that you] understand your setup and how to use DOS."

This paper was presented at the Aston conference as a workshop, which relieved the author of the responsibility of providing definitive answers. Most or all of what it contains may not be new to the reader, but some of it may be rather shocking since it may remind you what you have forgotten.

It deals with a topic that I consider important, because unless we can persuade more people to move from software that is marketed as 'easy to use' (whether it is or not!), (LA)TEX users run the risk of being marginalized — being seen as eccentric, pedantic, weird, 'out of their tree'.

The people. The people gathered at a TUG event will, for the most part, be the frontiersmen (sorry, persons — but I'll stick with the old word and the image) of TEX. They are also likely to over-represent academic users who have the benefit of support over the world-wide Internet. Furthermore, they include people like myself whose jobs are specifically to evaluate, implement and support software for an organization. We can justify the investment of time playing around — or research as we call it on Sundays. Many TUG members would describe themselves as 'computer scientists'. When I first obtained TEX 'for the University of Hull' I was startled to find *several* copies already on the system, fetched and installed by computer scientists. But their brief does not require them to provide services to others.

The frontiersmen are friendly and hospitable but they may have forgotten the Herculean efforts they endured to get where they are now. Their latest discoveries may look like wild flights of fancy to a newly arrived 'immigrant' used to the pedestrian disciplines of an office-oriented word processor.

British common law is largely based on what is expected of the 'reasonable man' (or woman). What is reasonable to ask of a putative TEX-installer? The following recent exchange on email with an archive maintainer is likely to send the vast majority of current computer users climbing up the wall, or up the tree that we vacated:

> A user asks, "We've been running TeX 3.0. I'd like to get the version 3.14 as on ymir.claremont.edu, but the files are a pain to transfer via ftp ...."
>
> And gets the reply, "If all you're after is TeX, all you should need is `TEX.WEB` and `TEX.VMS-CHANGES`, both of which are ASCII files. ... Also there and of interest are `VMS_TEX_NOTES.TXT` and `COMPILE_TEX.COM`.
>
> "Get the `WEB` and `VMS-CHANGES` files, then `TANGLE` (probably defined as the foreign command symbol `TANGLE:==$TEX_EXE:TANGLE` since you're running DECUS-TeX) the files, replying with the appropriate `.WEB` and `.VMS-CHANGES` files, also pointing to `TEX.PAS` as the output and `TEX.POO` as the Pool file. Once done, run Pascal on your `TEX.PAS`, then link `TEX.OBJ` and you should have the executable, `TEX.EXE`. Alternately, this is done for you in the compile DCL command script. To finish up, `COPY/PROT=W:RE TEX.EXE` to `TEX_EXE` and you should be done. The VMS-CHANGES file includes the `TEX.CLD` file which is distributed with the DECUS files, so the change should be transparent."

Let me stress I'm not criticising the work of archive maintainers. They do a superb job unpaid, and are greatly appreciated by many users including myself. However, TEX may be seen from outside as a club for computer experts, and advice like the above will repel more users than it attracts.

The program. TEX represents an attitude to both typography and computing. It is exciting and innovative. It is iconoclastic, dismissing arbitrary restrictions that are incidental to either the hardware or the system in use. For example, in Chapter 4 of *Computers in Typesetting* (Knuth, 1984), Knuth writes "This is something that authors and readers aren't accustomed to, because printers couldn't do such things with traditional lead types." It is therefore something of a *volte face* by Don Knuth to wish to embalm the TEX program and take it into the afterlife unchanged, or at least to ask that its name shall live for ever and ever. TEX *as used* is constantly evolving and developing, and this represents a problem for anyone who wants to get started.

Knuth's mandate relates only (I think) to the program source of TEX. Any usable 'TEX' system has to be built of many tools and components which are available in a variety of forms and from various sources. While 'mix and match' gives great flexibility and power, it also gives uncertainty and instability. One aspect of 'client-server' computer systems is the split between the front-end that the user sees and the processing engine. Papers at this and previous conferences have described many preprocessors and integrated front-ends that retain TEX as the background processor; there seems almost a fatalism that SGML will be adopted as a document-description language but that its input will be processed into TEX for display.

Most word processors look like Barbie® Dolls. TEX looks like a Meccano® set. The difference is that the doll is a toy with a limited use, and is heavily promoted because buying the doll leads to buying clothes and accessories — like boyfriend Ken. Incidentally, Barbie is anatomically distorted to hyperstimulate human emotional responses — rather like most software hype. Meccano®, for those who don't know it, comes as a kit of metal strips and plates which the user bolts together to build models. The sales pitch is that you are limited only by your imagination and perseverence. You are advised to start with something easy and warned that you may become an enthusiast.

Once again, I'm not criticizing or complaining; I am pointing to the appearance and suggesting that this attracts a certain type of clientele.

The mission. To put the problem into perspective, most programs (for personal computers) arrive as a package of disks and a manual. In the manual is a page or chapter called 'Installation' and usually the procedure involves nothing more than putting 'Disk Number 1' in the drive and typing `install`. Some programs even make changes to your system configuration (I don't like this feature) or indicate possible optimizations. In contrast, TEX looks like, for example,

- 150+ files for PCs from an archive, most of which are compressed, packed sets of files. This includes multiple versions of most programs, numerous, very technical, `readme` files, and complex environment and configuration variables, or

- an archive file for Unix that expands into 3500+ files in a hierarchy of directories that may not be allowed (by 'Systems') on the target system, or
- the TEX archives themselves, which contain (June 1993) 26000+ files of all vintages and provenances.

While checking this paper I found that a program called `install` has appeared in the emTEX archive; I have not yet tested this feature.

The TEX community, if it is to thrive, must provide guidance for two distinct people (though they might be in one body): the TEX implementor and the user. Both would benefit from having a defined canonical or default 'Current-TeX'. (I defer to Don Knuth on the use of the trademark.) Anyone wanting TEX should be able to get — easily — this definition and obtain the software in a form where it installs as easily as any other product. That is, they don't need to go to several sources, look for FAQs round the world, decypher cryptic in-jokes or take a course in systems programming.

In current (perhaps just going unfashionable) jargon, fellow TEXies, you have a marketing and image problem.

Moving to a solution

For an individual user, needing one copy of the software, I have no hesitation in recommending they go to a commercial supplier for a TEX package. That's how I got started. The first TEX I used was TurboTEX® and that came as a set of disks and a booklet. Nevertheless, there was considerable room for improvement to match the 'professional', i.e., slick, appearance of other software. Whether the product can be made sufficiently appealing to a mass audience is a commercial decision for the companies.

The corporate user, especially in the impoverished academic setting, has a greater problem. Commercial licences for the packaging make TEX an expense comparable with other word processing software. Advertising and peer-group pressure will incline the naïve (it's compulsory to include *this* word in any TEX document) users towards to most palatable at first sight. Academic software-support staff, unless they are inspired to claw through the undergrowth and fight up the rapids to join the frontiersmen, need far more help in installing the software and writing the mythical *Local Guide* than is currently available.

The user group, and especially the management committee, has a clear rôle to define the 'Current-TeX' standard and to keep an eye on developments so that the standard can be amended or extended by definitive statements, rather than by uncoordinated natural selection. As a start, this would allow archives to be divided in a stable kernel and a seething caldron of experimentation where files can be added *and deleted* in contrast to the current ever accumulating museum of endeavour.

The announcement of the CTAN — the Comprehensive TEX Archive Network (Greenwade, 1993) — at the Aston conference suggests that this is where 'Current-TeX' should be stored. From conference discussions it appears that this should be classified in at least two dimensions:

- as a series of workbenches with appropriate tools — for authors, editors, compositors, style designers, font designers, systems support, etc.
- as a set of levels — entry (absolutely required), advanced, expert ....

As a first stage the standard should be defined in terms of the products and versions that should be available at all installations. This information would be a single document or file. The next step would be to expand this list in terms of actual filenames for particular implementations. This could be done by setting up actual directories with copies of files, but more probably by storing the list of filenames as a document. Many text files would be common to all implementations, while executable binaries would be unique. There is the small but important point of ensuring that files that constitute 'Current-TeX' should not be deleted or changed except deliberately and in concert with changes to the filelist files.

A 'Current-TeX' document would assist site support staff like myself or commercial enterprises who want to sell TEX with 'added value'. At present, the contents do not seem to have changed since *The TEXbook* and *The LATEX User Guide* were written — around 1984. Is the 'new LATEX' (in archives as `latex.tex` dated 25 March 1992) now the standard? It's required for building the NFSS (Mittelbach, 1990) — is *that* now standard?

Defining a kernel

I am not here to prescribe. The following list was compiled from several years of supporting a generally-used system and from reading e-mail discussions. The TEX community as a whole must define what should be included. I'll start the process with a list of items, an indication of the files involved and a comment on why they should or should not be included. I've used DOS filenames since the extension codes (`.tex`, `.exe`, etc.) conveniently distinguish the type of files.

In some areas, e.g., editors, it is clear that 'Current-TeX' can specify that there must be *an* editor to allow users to manipulate ASCII files. The CTAN can make public domain programs available, but we cannot prescribe which the user will choose.

- TeX
  - `tex.exe`
  - `plain.fmt`
  - `tex.poo`

  TeX as supplied is so overwhelming that it's hard to relate this to the single program described in *The TeXbook*. For most widely-used machines, it would not be hard to maintain a single executable file that needs only copying and running. Barring dramatic problems, this could be updated annually. On the 'level of user' scale, I suggest that the great majority of users will want only the executables, in a form that they can copy and run. Reading and changing the source code is a mark of being 'expert'. BTW: I've never read *TeX, the Program* (Knuth, 1986).

  I was recently asked, "When the archive people talk about TeX 3.1, 3.14, ..., etc., does this refer to changes in the `.EXE` program or the `.FMT` or both?" I confessed that I didn't know. (I think it covers both, but even when changes are made only to `tex.exe`, it is necessary to reprocess `plain.tex` → `plain.fmt`.)

  In several years of using TeX, I've never had to make any changes to the `.POO` file, nor have I heard of anyone doing so. Is this a neglected facility for enhancing one's TeX, or a red-herring?
- LaTeX
  - `lplain.fmt`, incorporating
    1. `lplain.tex`
    2. `lfonts.tex`
    3. `latex.tex`
    4. `hyphen.tex` (via `lhyphen.tex`)
  - Standard styles:
    book (`book.sty`, `bk10.sty`),
    report, article, letter, ...
  - Many options:
    page sizes (with samples for parameter names and values), `multicolumn`, `fancyheadings` (should be the default), marginal notes, line numbering, etc.
  - Seminar (let's scrap SLITEX)

  It took me a long time to realize the relationship between `lplain.tex` and `latex.tex`. I still don't know where to look up such information. The layered design of LaTeX (and flaws in same) were discussed at a meeting of UKTUG in Oxford (February 1993). It would be silly to preempt the LaTeX 3 project, but there may be other styles that are widely enough used to be 'canonized'.

  As far as I know, LaTeX for most systems is still distributed with what is now called the 'old format', and each user has to rebuild the `.fmt` file to the 'new' LaTeX dated 25 March 1992 in archives. This is a major potential for problems and inability to exchange documents freely. The *LaTeX User Guide* (Lamport, 1986) has a disclaimer on the publisher's page, "Any discrepancy between this description and the behavior of this or any later release of Version 2.09 is an error." Ah, but in the documentation or the program?

  I think any supporter of a TeX system has to be able to build a format using `initex` (see below). What should they then do to verify that the new format is conformant? We can assume that they will not take kindly to the suggestion of running the `TRIP` test.

  Could the font size options be parameterized? This is more a question for the LaTeX 3 project, but the increasing use of arbitrarily scalable fonts (in PostScript) makes TeX's few fixed sizes look old fashioned.

  The standard paper size options should cover at least the range offered in DVIPS, and should probably incorporate 'best practice'. For example, I'd welcome some advice on good page designs for using A4 paper.
- Writer's tools
  - Simple ASCII editor
  - TeX-aware editor (e.g., emacs, TE — but without the crashes described below)
  - Spell checker
  - TeX-aware syntax checker
  - Word count and other 'editor's tools'

  We know TeX is not a word processor, but it is becoming increasingly difficult to buy a simple text editor that is not a word processor. This unfortunately is a double disincentive for many users to consider TeX, firstly as the word processor apparently does what they want and secondly because you have to actively avoid word processor features. You generally have to take special steps to save WP output as simple ASCII.

TUG could take steps to encourage marketing of commercial and other editors. I know it already does but I get the impression that the typical active TUG member is happy with an environment as complex as `emacs`, which would have most of my university users scrabbling again up that tree. TUG endorsement would add some official weight to complaints such as the two below.

During the Aston conference the editor `redit` became available on the CTAN. This looks attractive but the version available was only partly translated from German. This was actually useful, as running a German editor made it easier (as a non German-speaker) to focus on the structure and appearance.

The amSpell (no connection with AMS) program by Erik Frambach et al. (Frambach, 1992) is a public domain spell checker.

The TeXshell program by Jürgen Schlegelmilch (1993) provides a uniform interface to the various components of a TEX system on a PC. It still requires the local implementor to understand how to integrate each component into the whole.

Two recent problems that have involved me personally in this area have been:

1. the TE editor that can be obtained from archives completely hangs our PCs if told to edit a file greater than 60K. I have not been able to find support or the source of the program
2. I have been promoting the use of Correct Grammar (CG) as a *prompt* to encourage thinking about writing. The program is not TEX-aware and when I contacted the vendors I was told there were no plans to add this feature.

- Tools to build a format file — e.g., for alternative languages

  The ten and a halfth commandment reads, "This is the plain TeX format ... And don't modify the file under any circumstances." I put it as ten and a halfth because the better known eleventh commandment takes precedence. The most obvious area where people seem to want to override the contents of `plain` is in font allocation, and in adapting TEX to non-American languages. These are both areas where the average user does *not* want to do the spade-work, but would appreciate being able to be supplied with either simple instructions or, best of all, alternative format files for immediate use. The German TEX supplied with emTEX is perhaps a model for this. There is a TUG working group looking at language problems: perhaps formats for different languages should form part of 'Current-TeX', with English being just one amongst many.

- Fonts — as sets of characters
  - **TEX and LATEX:** Possibly the most FFAQ on fonts is, "Where is `lcircle10`?" TEX fonts are currently in a state of flux caused by the informal nature of TEX evolution. TUG members assume that the DC fonts are now 'standard'; TEX users outside TUG probably would ask, "What's DC?" At Aston it was clear that the only fonts that could reliably be expected in a currently 'standard' installation are those defined by Knuth (1984) and Lamport (1986).
  - **AMSFonts/:**
    Even those not writing mathematics may require extra symbols. The AMSFonts/ provide some 200 symbols not in the (LA)TEX sequences. I was made aware of this when a student asked for the three-dot 'therefore' symbol.
  - **NFSS:** As with DC, inside TUG and among users attached to the network and able to download from archives, NFSS is now the standard. Elsewhere?
  - **Viewing tools:** We need to be able to look at and to interrogate fonts, to have a simple way of searching a set of font files for a particular glyph by name. There are `testfont` and a similar macro by Borde (Borde, 1992), but that's it.
  - Tools for converting fonts
    (e.g., Adobe Type 1 → TEX).

- Fonts — as system files
  `*.TFM, *.PK, *.VF`

  METAFONT is almost certainly needed to avoid trying to store every font at every size. The local supporter maybe only needs to know how to run METAFONT with supplied input, how to handle the results and where to put the output.

  Users are confused and intrigued by the relationship between the printer resolution, design size and character magnification, and the bitmap file. I had to attend a TUG workshop to be clear on this. The documentation on virtual fonts is virtually restricted to the sources of VPtoVF and VFtoVP. The possibility of `.AFM` and character mapping fonts for Adobe® fonts also leaves the user bemused.

emTEX's system for packing font bitmaps into library files is commendable, but this leads to yet another extension (`.FLI`) and the need for tools to maintain libraries. A standard for porting fonts would be useful; as binary files they are a pain.

- Printer driver
  There is a lack of information about printer modes, but perhaps this is very technical and esoteric. However, it would be helpful to see some examples of how much (or little) improvement can be achieved by tweaking the parameters. I have installed Mattes' `dvidot` and Rokicki's `dvips` (Rokicki, 1993).
- Previewer
  A fast and versatile screen previewer can make TEX competitive with any WYSIWYG DTP system. Many people (meaning my own prejudice but reinforced by numerous discussions) have indeed commented that immediate design on the screen is less desirable because users who are not professional designers will both see what they expect and accept what they are offered. `dviscr`, `dviscreen` and `xdvi` are all very good.
- PostScript
  PostScript appears to be a world standard for page description as well as a printer driver. At least that's the impression I get, though it's hard to be objective. Some of my colleagues have suggested buying non-PostScript laser printers, and cite HPGL as an equally valid standard. However, in the TEX world, PostScript and Adobe font representations seem a centre of interest. I make heavy use of:
  - PSNFSS
  - PStricks

  (through `seminar.sty`, not as direct calls)
  - Graphics inclusion

  where `bbfig`, the PostScript code that checks `BoundingBox` values is an indispensible tool. I have installed on our system all of the `epsf` macros that come with `dvips` (Rokicki, 1993), the `BoxedEPSF` macros (Siebenmann, 1991) and the `psfig` macros (Darrell, 1987). None is definitive or clearly best.
  - PostScript previewer

  i.e., GhostScript — I use it heavily on DOS and UNIX
- Page manipulating tools
  - dvi manipulation
  - PostScript manipulation

  `dvidvi`, `dviselect`, `dvibook` and their `ps` equivalents
- Formats — on top of `plain` TEX
  - **Eplain:** Expanded Plain TEX (Berry, 1993). Provides tools for cross-referencing and indexing — i.e., LATEX-like facilities without the imposed styles.
  - **Newsletr:** `plain` TEX macros for multi-column working (Goatley, 1991).
  - **TEX by Example** (Borde, 1992).
  - **$\mathcal{A}\mathcal{M}\mathcal{S}$-TEX:** American Mathematical Society style (Spivak, 1986). This is an *alternative* to `plain`, so $\mathcal{A}\mathcal{M}\mathcal{S}$-LATEX/ may now be preferred.
  - **texinfo:** Gnu documentation format. Apart from being a format in its own right, this is a stepping stone to a wealth of other excellent software, some of which is TEX-related.
- Formats — on top of LATEX
  - **docstrip:** Another TUG de facto standard, for the distribution of 'documented' LATEX styles. George Greenwade describes it (in email, 1993) as "Literate programming for LATEX."
  - **$\mathcal{A}\mathcal{M}\mathcal{S}$-LATEX/:** American Mathematical Society options which is an extension standard LATEX. Requires NFSS.
- Easy options for verbatim file insertion, insertion of extracts from text files, etc.
- Pre- and post-processing tools
  - **Bibliographic tools:** BIBTEX and Tib (Alexander, 1989)
  - **Indexing:** MakeIndex
  - **Listing:** specific types of source documents (e.g., c2latex)

Whither?

Preparing this review has reminded me how many sources have been tapped to create the TEX system I currently maintain at the University of Hull. Thanks to the many people who have helped me, directly or indirectly.

This paper is an invitation and a challenge to comment to me, to TUG or to the wider TEX community. Even the isolated user with a personal copy of TEX will at some time want to send a document to another TEX user or will want to upgrade their system. Either of these events will cause unnecessary anguish if they are aiming at a wobbly target.

Another necessary aspect of 'Current-TeX' will be consistent and accessible documentation. One way forward will be to gather *Local Guides* from many sites, to compare what is made available and how it is accessed. It should be a fair assumption that *Local Guides* are themselves written in (LA)TEX. If you have written a document, please deposit a copy with the CTAN.

What's an archive?

The provisional recommendations here contain several items that are commonly distributed through the TEX world-wide archive network. These are then retrieved *in their instantaneous incarnation* by file transfer or email. If the 'Current-TeX' concept is taken up by TUG, this is not a satisfactory way of providing a consistent system for standard sites. From discussions at Aston, it seems that those who maintain archives are sympathetic to the views expressed in this paper. Various technical methods for keeping archives internally consistent were suggested and will be further discussed.

Bibliography

Alexander, James C. "**Tib** A TEX Bibliographic Preprocessor. Version 2.2." Public domain software available from TEX archives. 1989.

American Mathematical Society. "AMS-LATEX/ Version 1.0 User's Guide." 1990.

Berry, Karl. "Expanded Plain TEX — Version 2.3." Public domain software available from TEX archives. March 1993.

Borde, Arvind. *TEX by Example: A Beginner's Guide.* Academic Press, 1992.

Childs, Bart, et al. "Syllabi for TEX and METAFONT Courses." *TEXniques* 7, Pages 117-128, 1988.

Daneliuk, Tim. "List of TEX-Related Tutorials as of 05.18.93." email available from `info-tex@shsu.edu`, 1993.

Darrell, Trevor. "`psfig` Version 1.8." Public domain software available from TEX archives, 1987.

Doob, Michael. *Gentle Introduction to TEX.* Public domain document available from TEX archives, or printed form from the American Mathematical Society, 1992.

Frambach, Erik, et al. "amSpell — Version 2.03." Public domain software available from TEX archives, 1992.

Greenwade, G. "The Comprehensive TEX Archive Network." *TUGboat* **14**(3), in press, 1993.

Goatley, Hunter. "`NEWSLETR` — `plain` TEX Macros for Newsletters." Public domain software available from TEX archives, 1991.

Knuth, Donald E. *The TEXbook.* Volume A in *Computers and Typesetting.* Reading, Mass.: Addison-Wesley, 1984.

Knuth, Donald E. *TEX, the Program.* Volume B in *Computers and Typesetting.* Reading, Mass.: Addison-Wesley, 1986.

Lamport, L. *LATEX: A Document Preparation System.* Reading, Mass.: Addison-Wesley, 1986.

Luecking, Don. "Setting up emTEX." Information file available from TEX archives, 1993.

Maltby, Gavin. *An Introduction to TEX and Friends.* Public domain document available from TEX archives, 1992.

Martin, C. R. "TEX for TEXnical Typists." *TUGboat* 11 No 3, Pages 425-428, 1990.

Mattes, Eberhardt. "emTEX." Public domain document available from TEX archives, 1993.

Mittelbach, Frank, and Rainer Schöpf. "The New Font Family Selection." *TUGboat* 11 No 1, 1990.

Rahtz, Sebastian. "A LATEX Survival Guide." Notes distributed with Sun Sparc implementation of TEX, 1992a.

Rahtz, Sebastian. "System Guide for Setting Up TEX on a Sparc." Notes distributed with Sun Sparc implementation of TEX, 1992b.

Rokicki, Tomas. "DVIPS 5.519" Public domain software available from TEX archives, 1993.

Schlegelmilch, Jürgen. "TEXShell — Version 2.5.2." Public domain software available from TEX archives, May 1993.

Siebenmann, Laurent, "`BoxedEPSF.TEX`." Public domain software available from TEX archives, May 1991.

Spivak, Michael. *The Joy of TEX.* Addison-Wesley, 1986.

Warbrick, Jon. "Essential LATEX." Public domain document available from TEX archives, 1992.

Van Zandt, Timothy. "PSTricks — PostScript Macros for Generic TEX." Public domain software available from TEX archives, 1992.

Van Zandt, Timothy. "`seminar.sty` — A LATEX Style for Slides and Notes." Public domain software available from TEX archives, 1992.

The Comprehensive TEX Archive Network (CTAN)

George D. Greenwade
Department of Economics and Business Analysis
College of Business Administration
Sam Houston State University
Huntsville, TX, USA 77341-2118
Voice: (409) 294-1265
FAX: (409) 294-3612
Internet: `bed_gdg@SHSU.edu`

Abstract

This paper outlines the concept, development, and use of the Comprehensive TEX Archive Network (CTAN) — a network-accessible archive for files related to the TEX family of document processing. The CTAN is a coordinated effort among consenting well-known archive sites which provides quick identification and retrieval files in a consistent manner from hosts on different continents, thereby reducing overall network load and increasing speed of retrieval. Moreover, it provides users with a parallel archive structure between hosts with holdings which are generally synchronized to within 30 hours of one another. This is achieved by routinely mirroring one another's holdings, as well as mirroring other archives to maintain an up-to-date collection of files.

Why a Comprehensive TEX Archive Network?

Since the inception of publicly-accessible network-based archives, TEX and its related packages, macros, and utilities have been available for retrieval by users via any number of techniques. The combination of the growth of the Internet in recent years, the growth of publicly-accessible network-based archive sites, and the growth in the number of files associated with TEX and it's affiliated packages and programs, created a rather overwhelming number of files for users with network connections to sort through. In terms of "overwhelming," the number of files available has been a significant boost for users; however, in these same terms, the number of different versions available, their precise location on a given archive host, the user interface available to access these files, and the ability to efficiently identify the various pieces required to make the various iterations of TEX and its relatives work properly has evolved to be a non-trivial task.

In recognition of these problems, the then-newly-created Technical Council of the TEX Users Group formed a Technical Working Group on TEX Archive Guidelines (officially WG-92-05; informally referred to as TWG-TAG) in the latter months of 1992, with the author as its Chair.[1] While a variety of issues related to archiving have been discussed (and broader guidelines, *per se*, will in all likelihood be forthcoming), the concept of creating a systematically coordinated family of network-based archive sites was tacitly agreed upon as a mechanism for verifying that the ideas under discussion were workable. Also, this approach was viewed as a mechanism for creating a more efficient design, both theoretically and practically, to meet the needs of the worldwide TEX community.

General Consensus Notes From TWG-TAG. Germane to the development of a set of archives in lieu of a "hard and fast" set of guidelines for propagation to other hosts were the following concerns:

[1] I would like to take this opportunity to formally recognize the members of this Working Group and publicly thank them for their efforts on any number of topics which we have dealt with. These individuals are, in alphabetical order: Nelson Beebe, Barbara Beeton, Karl Berry, Johannes L. Braams, David Carlisle, Michael J. Ferguson, Alan J. Hoenig, Don Hosek, David M. Jones, Pierre MacKay, David Osborne, Philip Taylor, Jon Radel, Sebastian Rahtz, Rainer Schoepf, Joachim Schrod, and Elizabeth Tachikawa.

- Existing archive hosts very likely have chosen and utilize a structure which is politically and practically acceptable for their site.
- While the concept of the Internet's File Transfer Protocol (ftp) for a user interface was on everyone's mind, access via alternate means (primarily electronic mail and hard media) had to be considered.
- Very few hosts on the network possess a comprehensive archive of TEX-related materials; thus, a design guideline which includes all dimensions of TEX may not be proper for a specialized archive.
- Even if a set of guidelines were developed, there is no way to ensure that every site which possesses some aspect of TEX in its archive would follow them since archives are a function of local resources, support, and needs more than "network" demands.
- If a workable demonstration of the guidelines existed, more sites may voluntarily elect to follow the ultimate guidelines.
- No single site on the network possessed a canonical listing, much less collection, of the latest relevant files available; therefore, any guidelines developed would be hypothetical more than working.
- It is necessary to make the structure as flexible as possible, while at the same time ensuring that files may be easily located by users via some consistent and logical design.
- As much as foreseeably possible, the structure should be extensible into evolving network retrieval and browsing technologies, such as Gopher, Mosaic, and other developing utilities.
- It is essential that the archives support users from the variety of platforms under which TEX is available; while a given platform may be used for the archives themselves, it should not impose problems on any other platform which a user may wish to ultimately use the files on.
- At least initially, the concern was to provide a reliable archiving directory hierarchy from within the project, as opposed to a production system hierarchy.[2]

[2] Please note that the concerns above are those perceived by the author as the majority opinion and are not necessarily those of the TWG-TAG, nor are they necessarily those of any one given member of the TWG-TAG.

Genesis of the Design

Prior to defining the CTAN directory structure, extensive discussion was undertaken by TWG-TAG regarding the optimal directory hierarchy. Compared for use were the hierarchical design used on most ftp-accessible hosts now possessing a TEX archive and the flat design used by a number of mail-oriented hosts. For brevity's sake, allow it to be said that the hierarchical design was chosen. Once this decision was made, comparisons of existing archives were undertaken, primarily focusing on the holdings of three major sites — Stuttgart University's `ftp.uni-stuttgart.de`, Aston University's `TeX.ac.uk`, and Claremont College's `ymir.claremont.edu`. The end result is a hybrid of the three directory structures, focusing on top-level directories which are somewhat mnemonically-based directory names at this level which, at least in the author's view, is pleasing.

The structure is adequately diverse so that dvi-related files (such as device drivers, dvi to output programs, and dvi viewers) are distinctly different from macros, and that macros are adequately categorized into the appropriate flavor of TEX for which they are intended. The top-level directory hierarchy is presented in Appendix A.

Conceptually, every file in the archive fits into one branch of this directory hierarchy (albeit the directory mutually agreed to by the maintainers of the hosts involved in this project). Where a file conceptually fits into more than one directory, efforts are made to ensure that the file properly appears where it should.

Coincidences of Consequence. A critical dimension of this project was its timing. Two hosts were new machines; therefore, they were easily designed into whatever structure was agreed to. The main coincidences which aided the success of this project were:

- Sebastian Rahtz, who was just beginning to put together the Aston "daughter" archive at `ftp.tex.ac.uk`, began utilizing the directory design by following the very rough outline of the preliminary structure. Making the structure operational was a significant factor in more than a few subsequent decisions as it illustrated, in a close-to-production environment, the strengths and weaknesses of the then-tentative hierarchy.
- Sam Houston State University was just installing its first Unix-based host and learning its idiosyncrasies; however, its use as an

archive host was established soon after its installation. This was a preferable choice for an archive host over SHSU's more established ftp host, `Niord.SHSU.edu`, as it ran a comparable software to the other two hosts, and was configurable for use as a mirror.

- David M. Jones had just released the first version of his index of TeX and LaTeX macros, providing the TWG-TAG with a relatively comprehensive listing of files, their authoritative location, current version, and other critical information.
- Joachim Schrod's mirroring software for Unix hosts (which the CTAN hosts utilize) was being upgraded. Joachim was able to quickly address a few specific problems resulting from its massive use.
- Nelson Beebe was developing a few new extensions to the ftp server software which was adopted for use. Significant contributions were his definitions which allow users to retrieve recursive ZIP and ZOO archives of directory holdings.
- A bug was located in the adopted ftp server software. This led to the new release of the ftp server software, resulting in a much nicer interface, easier management, and more generic installation as compared to prior versions. In essence, while each host is unique in architecture, the configurations are virtually parallel, as are most features available to users.
- Karl Berry further extended his somewhat standardized Unix TeX directories and installation. While this has yet to play a significant role in design, it provides significant future extensions to the services of the CTAN.
- The University of Minnesota upgraded the facilities and functions of its `Gopher` software. This interface now serves as an alternative access method to the CTAN at Aston and SHSU.
- Significant enhancements to the ZIP utilities by the Info-ZIP team were released. This is a major item of concern as it allows for a platform-independent mechanism for the archival of files held within the CTAN collection. In essence, when combined with Nelson Beebe's extensions, this feature allows users on virtually any platform to retrieve recursive directory holdings, then convert them to the operating system specific needs which they confront.

By no means is this a complete listing of all coincidences of consequence to the project — suffice it to say that without a number of apparently disjoint and unrelated projects coming to fruition at approximately the same time, this project would likely still be in its planning stages.

The CTAN Hosts

By now, you're very likely asking where these hosts are on the network. The present CTAN hosts and their root CTAN directories are:

```
ftp.uni-stuttgart.de   /pub/tex/
ftp.tex.ac.uk          /pub/archive/
ftp.shsu.edu           /tex-archive/
```

Based at Stuttgart University (Germany),[3] Aston University (England),[4] and Sam Houston State University (United States),[5] respectively.

While the designs of the consulted archives were largely congruent, there were some modifications when compared to all existing archives. Rainier Schoepf, TeX archive manager at Stuttgart's `ftp.uni-stuttgart.de`, was fully agreeable to modifying the directory structure there based on the recommendations which were developed. While it was clear that the changes would impose short run problems for users of his collection and potential problems in the management of his site's mirroring of packages, he was in agreement that the long term benefits to the TeX community of a well-known and specified structure outweighed any short term impact. Also, as the other two hosts involved were new to the network, being able to plan for the inclusion of an existing large and active archive from the start was a real benefit, especially since it had a mirroring structure already in place and the site was mirrorable by other sites.

[3] In addition to the administration of the institution, thanks are extended to DANTE, the German speaking TeX Users Group, for their archival support and leadership role in maintaining this archive.

[4] In addition to the administration of the institution, thanks are extended to the UK TeX Users Group for their archival support and leadership role in the development of their original archive, as well as in the creation and development of this new archive host.

[5] In addition to the administration of the institution, thanks are extended to SHSU's Computer Services Division, which has been more than forthcoming in support and access as this archive has evolved. Additionally (and especially), thanks are extended to SHSU's College of Business Administration for its continued interest in and support of this project.

What's Available

As noted in the name selected for these hosts, the *Comprehensive* TEX Archive Network, the collections available are about as comprehensive and timely as humanly possible. Do these hosts have "everything"? The truthful answer is that they probably do not and probably never will — but they are and will be about as comprehensive a collection as can be arranged. As sites possessing relevant files and packages are identified, one of the CTAN hosts will include them in its mirroring passes; from there, the files will then propagate to the other CTAN hosts. Presently, there are about 790 megabytes of files available in the `/tex-archive/` directory tree on `ftp.shsu.edu`. It would be expected that the collections in the TEX archive area on the other two hosts would be virtually identical.

How It Is Achieved. The synchronization between hosts is handled within a "mirroring" program whereby files on one host are propagated to the other hosts of the CTAN. The variant of mirroring used by the CTAN hosts is a modification of a family of perl scripts written by Joachim Schrod.[6]

Conceptually, the process entails three major points. First, hosts where files are authoritatively available for retrieval are mirrored inbound by a selected CTAN host. This CTAN host locates these files in a pre-specified area of the collection. Second, users may contribute files to a given CTAN host by uploading their files into an "incoming" area on any given CTAN host. From there, the local CTAN administrator moves these contributions into the CTAN hierarchy. Finally, the CTAN hosts mirror one another on a routine basis to collect all new files which have entered the collection at the mirrored host. In all, about one gigabit of files are referenced daily by the mirroring processes in order to maintain accuracy, timeliness, and correctness of holdings.

Existing User Interfaces

As noted throughout this paper, the Internet's File Transfer Protocol (ftp) is supported at the moment for a user interface. There are three other interfaces which should be recognized — Gopher, mail, and hard copy — as these are important to users which prefer to utilize them.

[6] The program itself is available in the `archive-tools/mirror/` directory of the CTAN hosts.

ftp. The ftp interface utilized by each CTAN host is the "wuarchive `ftpd`" program.[7] In addition to the normal ftp commands a user would expect, these hosts support a variety of additional useful functions, such as locating a file, creating `ZIP`, `ZOO`, and Unix `tar` archives on-the-fly, automatic compression of files on-the-fly, and a few other features. As these features may be modified, the best policy is to consult any files named `README` in the default directory at login.

Gopher. Two of the CTAN hosts, `ftp.tex.ac.uk` and `ftp.shsu.edu` are configured for Gopher access. The Internet Gopher, developed by the University of Minnesota, provides users with a menu-driven interface for transactions. Users may view and retrieve files in a fashion somewhat more friendly than ftp. Also, the Gopher menus presented do not necessarily represent files actually available at a site; instead, the client/server relationship utilized allows for servers to point to information stored elsewhere, which users are automatically connected to by simply selecting a menu entry.[8]

Pointing the Gopher client directly to the Gopher server on `ftp.tex.ac.uk` is the preferred method for access to the Aston CTAN archive; Gophering to SHSU's "front door" Gopher server on `Niord.SHSU.edu` is the preferred method to access SHSU's CTAN archive. In addition to the archives themselves, a variety of TEX-related files and services are available on each host. Sample menus from each are provided in Appendix B.

Mail. At the moment, no agreed-to mail interface has been installed on all hosts. The `ftpmail` program will be used and this will be documented better in a future article scheduled for *TEX and TUG News* discussing user interfaces to the CTAN hosts.[9]

Hard Copy. Sebastian Rahtz has already made arrangements and compiled a hard copy of the CTAN archives for CD-ROM distribution. The distribution

[7] This program is available in the directory `archive-tools/ftpd/mirror/` on the CTAN hosts. It is the classic ftp server/daemon utilized by Washington University in St. Louis, Missouri (USA) for their massive collection of files.

[8] Gopher clients are available for a wide variety of platforms. Sources for most platforms are available in the directory `archive-tools/gopher/` on the CTAN hosts.

[9] Sources for the `ftpmail` service are available in the directory `archive-tools/ftpmail/` on the CTAN hosts.

is available from Prime Time Freeware. The first version of this file set is already available and it is to be routinely updated to reflect changes made to the archives.

Aspects for Authors

While the process of mirroring and uploading provides a rough and ready mechanism for achieving the reliability of the CTAN, authors should be aware of a few dimensions. At the moment, no specific guidelines exist on these topics, so please view the following as personal views which will, in time, be discussed.

File Headers and Checksums. Nelson Beebe has developed a package written as an Emacs Lisp file which creates "standardized" file headers.[10] These headers, while clearly consuming space within the archive, provide valuable information which users and archive maintainers may refer to quickly. Moreover, these file headers provide valuable indexing information which may be blended into existing indices, such as David Jones' heroic effort. An example file header taken from a contributed LaTeX style option is provided in Appendix C.

Optionally included in these headers is a "checksum" option, which is of significant benefit whenever files are transferred electronically to verify correctness. The preferred is Robert Solovay's CWEB-based program which, with the assistance of Nelson Beebe, has been ported to a number of operating systems.[11]

Location of Files. It would behoove authors who maintain authoritative file set(s) on network hosts which are ftp-accessible to contact me so that arrangements may be made to mirror these files into the CTAN collection. Without a doubt, the ability to mirror files into the CTAN hierarchy is the least painful and most efficient method available as human intervention arises mainly in the review of the logs of the mirroring session. If an authoritative host is identified, its holdings will automatically be included when the site is referenced. In this way, additional copying to additional hosts for propagation will not be needed. Moreover, you will ensure that the latest versions are available for public consumption.

Directory Suggestions. Although it is well-specified, no official designation of the CTAN directory structure now exists. However, it is advisable for maintainers of authoritative files and packages to consider the utilization of the CTAN hierarchy, if at all possible. In this manner, the author will be able to envision how the archive is laid out for the end user. Also, if this structure is used, especially on Unix-based hosts, proper linkage to related files may be used. At the moment, one of the most wasteful uses of the CTAN directory contents are the multiple copies of files which authors elect to include. For example, as of 1 July 1993, approximately 27 copies of the LaTeX macros exist in the archive — only one of which (in `macros/latex/distribs/`) is authoritative. With linking, the mirror simply calls in the link and the latest versions of related files (from their well-specified directory) will be delivered to users.

Getting a File into the CTAN Without Mirroring

If mirroring is not practical (either because of inability to mirror a host, lack of a public ftp area in which to place a file, lack of ftp altogether, the contribution is a small number of files, or any other reason), authors can still easily get files included into the CTAN with electronic mail or via anonymous ftp.

Electronic Mail. To submit a contribution via electronic mail, use the address:

`CTAN-Mgr@SHSU.edu`

or

`CTAN-Mgr@SHSU.BITNET`

including the file in whatever manner is feasible (encoding of executables, splitting of files, indicating within a single post with multiple files where to "cut", etc.). In these cases, checksums are ideal as it provides verification that the file received for archiving purposes is indeed the same file which the author intended for inclusion. A brief note describing the contribution is very appreciated. As soon as the contribution has been processed into the CTAN, the submitting author will be notified via return electronic mail.

Anonymous ftp. To submit a contribution via ftp, connect to one of the CTAN hosts with an ftp client. When prompted for a username, type `"anonymous"` (all lowercase; without quotes) and use your complete electronic mail address as your

[10] This package is available in the `archive-tools/filehdr/` directory of the CTAN hosts.

[11] This package is available in the `archive-tools/checksum/` directory of the CTAN hosts.

password. The Aston and SHSU CTAN hosts support the /incoming/ directory for contributions (uploads) of files; the Stuttgart CTAN host supports the /soft/incoming/tex/ directory for contributions of TEX-related files. Once connected, a typical upload session would look like:

```
cd <appropriate incoming area>  !! 1.
mkdir <your directory name>     !! 2.
cd <your directory name>        !! 3.
[binary | ascii]                !! 4.
[m]put your file(s)             !! 5.
```

ignoring everything beginning with and to the right of !! above. Each step represents:

1. cd — change directory to proper incoming area (see above).
2. mkdir — make subdirectory (optional), which is especially nice to retain together multiple files contributions which are intended to stay together; you may use any your directory name you please.
3. cd — change to the directory you just created so you may use it.
4. binary or ascii — just to be safe, it's always best to verify the intended transfer mode (binary or ascii) prior to transfer.
5. [m]put — use standard ftp put or mput command to place your files in the incoming area.

Future Directions

While the CTAN hierarchy and holdings between its hosts are relatively stable at present, this does not imply that the project is completed. Topics for potential extension are:

1. The creation of on-demand "kits". At present, the systems/ directory hierarchy includes sets of packages intended for system-specific installation. One very serious potential problem with these file sets is that they are not automatically updated when new files are introduced to the archive. For example, some of the system-specific installations include prior versions of macro sets (generally, the LATEX macros in these file sets are outdated), drivers (such as the rapidly changing dvips), or other dimensions of the included software. This implies that users retrieving these file sets, while able to install TEX, have to immediately return to the archives to upgrade their systems.

An alternative to the present process would be for servers to have the capability of creating on-demand up-to-the-minute "kits" for platform-specific installations. However, specific design features for this functionality must be developed.

2. The use of a non-ftp mirroring mechanism. At present, the mirroring mechanism used by the CTAN hosts suffice for its use. However, ftp is not an efficient mechanism for moving the quantity of files involved in this project. A much preferable solution would be the use of some alternate protocol which allows for verification and automatic updating of all hosts in the CTAN.

3. Addition of other CTAN hosts. At present, the workload of the three hosts involved in this project is non-trivial. Essentially, the eastern side of the Atlantic is served by two hosts, while North America is served by a single host. The present configuration of hosts lends itself to easy extension to another North American host to service the Americas, as well as a Pacific-based host to serve Asia, the Pacific Rim, and the Southern Hemisphere. Simply creating a mirror of one of the existing CTAN hosts in these regions, with no additional network responsibilities, would be more than acceptable.

4. Automation of archival information. While the CTAN possesses the most comprehensive collection of TEX-related materials, one very dissatisfying aspect remains. This problem area lies in the collation of information which is quickly retrievable (even by the archivists themselves) which points to related files, required files, version requirements, authoritative location of files, listing of most recently added files within an area, etc. The mirroring process is marvelous at collecting files; however, without proper and somewhat standardized documentation of each component of the archive, tracing problems may prove to be painful.

At present, David Jones' index of macros — a wholly volunteer effort — exists and serves this function admirably within the context of macros. Additionally, the "Frequently Asked Questions" (FAQ) files from the USENET newsgroup comp.text.tex provide more information on other selected aspects which are included in the collection. However, the fact that these are volunteer efforts, as well as the fact these only cover a microcosm at a point in time of the complete and rapidly changing archive, is troubling.

Appendix A

The Top-Level Directory Hierarchy

Once into the CTAN host's root area, the following directory hierarchy is presented:

```
archive-tools/
bibliography/
digests/
documentation/
dviware/
fonts/
graphics/
help/
indexing/
languages/
local/
macros/
misc/
support/
systems/
web/
```

In brief, the contents of these directories include:

`archive-tools/` contains the various archiving tools which users may find useful.

`bibliography/` contains bibliography-related files, such as BIBTEX.

`digests/` contains back issues of TEX-related periodicals.

`documentation/` contains files and tutorials which document various aspect of TEX.

`dviware/` contains the various dvi-to-whatever filters and drivers.

`fonts/` contains a collection of fonts, both sources and pre-compiled.

`graphics/` contains utilities and macros related to graphics.

`help/` contains files which provide an overview to the archive and the TEX system.

`indexing/` contains utilities and related files for indexing documents.

`languages/` contains non-English related implementations of TEX.

`local/` contains local site-specific files — not of general interest.

`macros/` contains macros for TEX and its derivatives in unique subdirectories.

`misc/` contains files and programs which cannot otherwise be catalogued.

`support/` contains files and programs which can be used in support of TEX.

`systems/` contains complete system setups, organized by operating system.

`web/` contains WEB-related files and utilities.

Appendix B

Gopher Menus from Aston and SHSU

The menus for the CTAN archives via Gopher as of the time of this paper are provided below. For `ftp.tex.ac.uk` the Gopher menu structure appears as:

```
The menu's for the CTAN archives via Gopher as of the time of this paper
                    Root gopher server: ftp.tex.ac.uk

           1.  UK TeX Archive/
           2.  Aston University/
           3.  Minnesota Gopher root/
           4.  Archaeology/
           5.  Archie <?>
           6.  Font Samples/
           7.  General WAIS databases/
           8.  Looking things at AMS <TEL>
           9.  Veronica (search menu items in most of GopherSpace)/
           10. World Wide Web (Hypertext fra CERN) <TEL>
```

and selecting item 1. from this menu yields:

```
                           UK TeX Archive

           1.  Welcome.
           2.  Archive directory/
           3.  Indexed FTP (Directories names only) of UK TeX Archive.
           4.  Indexed FTP of UK TeX Archive.
           5.  Indexed access to archive hierarchy <?>
           6.  Other Archives/
           7.  TeX Font Samples/
           8.  UK-TeX.
           9.  WAIS database: TeX --- list of FTP sites with TeX material <?>
           10. WAIS database: TeX Frequently Asked Questions (UK specific) <?>
           11. WAIS database: TeX index of styles and macros (by David Jones) <?>
           12. WAIS database: The TeX Book <?>
           13. WAIS database: back issues of TeXhax, 1986 -- <?>
           14. WAIS database: back issues of UKTeX newsletter <?>
```

Selecting item 2. from this menu yields the directory hierarchy described in Appendix A.

For Niord.SHSU.edu the Gopher menu structure appears as:

```
                    Root gopher server: Niord.SHSU.edu

      1.  About the Sam Houston State University Gopher.
      2.  Customizing the Print command at SHSU.
      3.  All the Gopher Servers in the World/
      4.  Chronicle of Higher Education "EVENTS in ACADEME"/
      5.  Economics (SHSU Network Access Initiative Project)/
      6.  Internet Information/
      7.  LaTeX3 Public Document Library/
      8.  Libraries, Periodicals, References, etc./
      9.  Minnesota Gopher (Mama Gopher; get *your own* clients here!)/
      10. SAMINFO -- Sam Houston State University Information System <TEL>
      11. TeX-related Materials/
      12. Thesaurus and Dictionaries/
      13. USENET News (from Oakland University)/
      14. VMS Gopher-related file library/
      15. Veronica (search menu items in most of GopherSpace) /
      16. Weather Forecasts (National Weather Service; US)/
      17. Weather Underground (University of Michigan) <TEL>
      18. anonymous ftp archives on ftp.shsu.edu/
```

and selecting item 11. from this menu yields:

```
                         TeX-related Materials

      1.  Comprehensive TeX Archive Network (CTAN) at SHSU/
      2.  UK TeX Archive/
      3.  Archives of INFO-TeX/ctt-Digest (comp.text.tex)/
      4.  EconBib (LaTeX/BibTeX styles for economics)/
      5.         .*.*. FAQs, REFERENCE and PRIMERS .*.*.
      6.  FAQ for comp.text.tex (text and WAIS indexed)/
      7.  FAQ Supplement for comp.text.tex (text and WAIS indexed)/
      8.  TeX-Index (text and WAIS indexed)/
      9.  FAQ for comp.fonts (text and WAIS indexed)/
      10. The Canonical list of MetaFont fonts.
      11. A Gentle Introduction to TeX
      12. Components of TeX/
      13. Essential LaTeX/
      14. MetaFont for Beginners.
      15. NFSS in the Context of LaTeX
      16.             .*.*. RELATED FILES .*.*.
      17. LaTeX3 Public Document Library/
      18. Literate Programming Library/
```

Selecting item 1. from this menu yields the directory hierarchy described in Appendix A.

Appendix C

A Sample " Standard " File Header

```
%%% ====================================================================
%%%  @LaTeX-style-file{
%%%     filename        = "showkeys.sty",
%%%     version         = "1.01",
%%%     date            = "25 August 1992",
%%%     time            = "11:32:08 BST",
%%%     author          = "David Carlisle",
%%%     address         = "Computer Science Department
%%%                        Manchester University
%%%                        Oxford Road
%%%                        Manchester
%%%                        England
%%%                        M13 9PL",
%%%     telephone       = "+44 61 275 6139",
%%%     FAX             = "+44 61 275 6236",
%%%     checksum        = "61501 431 1786 14304",
%%%     email           = "carlisle@cs.man.ac.uk (Internet)",
%%%     codetable       = "ISO/ASCII",
%%%     keywords        = "LaTeX, label, ref, citation, keys",
%%%     supported       = "yes",
%%%     docstring       = "
%%%
%%%     showkeys.sty
%%%
%%%     A LaTeX style option which causes
%%%     \label, \ref, \pageref, \cite and \bibitem
%%%     to print their argument for proof reading purposes. The main
%%%     feature of this style is that these labels are printed in such a
%%%     way as to minimise the changes caused to the formatting of the
%%%     rest of the document text.
%%%
%%%     Documentation requires Mittelbach's doc.sty.
%%%
%%%     The checksum field above was produced by
%%%     Robert Solovay's checksum utility.",
%%%  }
%%% ====================================================================
```

Institutional Members

The Aerospace Corporation, *El Segundo*, *California*

Air Force Institute of Technology, *Wright-Patterson AFB*, *Ohio*

American Mathematical Society, *Providence*, *Rhode Island*

ArborText, Inc., *Ann Arbor*, *Michigan*

ASCII Corporation, *Tokyo*, *Japan*

Brookhaven National Laboratory, *Upton*, *New York*

Brown University, *Providence*, *Rhode Island*

California Institute of Technology, *Pasadena*, *California*

Calvin College, *Grand Rapids*, *Michigan*

Carleton University, *Ottawa*, *Ontario*, *Canada*

Centre Inter-Régional de Calcul Électronique, CNRS, *Orsay*, *France*

CERN, *Geneva*, *Switzerland*

College Militaire Royal de Saint Jean, *St. Jean*, *Quebec*, *Canada*

College of William & Mary, Department of Computer Science, *Williamsburg*, *Virginia*

Communications Security Establishment, Department of National Defence, *Ottawa*, *Ontario*, *Canada*

Cornell University, Mathematics Department, *Ithaca*, *New York*

CSTUG, *Praha*, *Czech Republic*

E.S. Ingenieres Industriales, *Sevilla*, *Spain*

Elsevier Science Publishers B.V., *Amsterdam*, *The Netherlands*

European Southern Observatory, *Garching bei München*, *Germany*

Fermi National Accelerator Laboratory, *Batavia*, *Illinois*

Florida State University, Supercomputer Computations Research, *Tallahassee*, *Florida*

GKSS, Forschungszentrum Geesthacht GmbH, *Geesthacht*, *Germany*

Grinnell College, Computer Services, *Grinnell*, *Iowa*

Grumman Aerospace, Melbourne Systems Division, *Melbourne*, *Florida*

GTE Laboratories, *Waltham*, *Massachusetts*

Hungarian Academy of Sciences, Computer and Automation Institute, *Budapest*, *Hungary*

Institute for Advanced Study, *Princeton*, *New Jersey*

Institute for Defense Analyses, Communications Research Division, *Princeton*, *New Jersey*

Intevep S. A., *Caracas*, *Venezuela*

Iowa State University, *Ames*, *Iowa*

Los Alamos National Laboratory, University of California, *Los Alamos*, *New Mexico*

Louisiana State University, *Baton Rouge*, *Louisiana*

MacroSoft, *Warsaw*, *Poland*

Marquette University, Department of Mathematics, Statistics and Computer Science, *Milwaukee*, *Wisconsin*

Masaryk University, *Brno*, *Czechoslovakia*

Mathematical Reviews, American Mathematical Society, *Ann Arbor*, *Michigan*

Max Planck Institut für Mathematik, *Bonn*, *Germany*

National Research Council Canada, Computation Centre, *Ottawa*, *Ontario*, *Canada*

Naval Postgraduate School, *Monterey*, *California*

New York University, Academic Computing Facility, *New York*, *New York*

Nippon Telegraph & Telephone Corporation, Software Laboratories, *Tokyo*, *Japan*

Observatoire de Genève, Université de Genève, *Sauverny*, *Switzerland*

The Open University, Academic Computing Services, *Milton Keynes*, *England*

Personal TEX, Incorporated, *Mill Valley*, *California*

Politecnico di Torino, *Torino*, *Italy*

Princeton University, *Princeton*, *New Jersey*

Rogaland University, *Stavanger*, *Norway*

Ruhr Universität Bochum, Rechenzentrum, *Bochum*, *Germany*

Rutgers University, Computing Services, *Piscataway*, *New Jersey*

St. Albans School, *Mount St. Alban*, *Washington*, *D.C.*

Smithsonian Astrophysical Observatory, Computation Facility, *Cambridge*, *Massachusetts*

Space Telescope Science Institute, *Baltimore*, *Maryland*

Springer-Verlag, *Heidelberg*, *Germany*

Springer-Verlag New York, Inc., *New York*, *New York*

Stanford Linear Accelerator Center (SLAC), *Stanford*, *California*

Stanford University, Computer Science Department, *Stanford*, *California*

Texas A & M University,
Department of Computer Science,
College Station, Texas

United States Military Academy,
West Point, New York

Universität Augsburg,
Augsburg, Germany

University of British Columbia,
Computing Centre,
Vancouver, British Columbia, Canada

University of British Columbia,
Mathematics Department,
Vancouver, British Columbia, Canada

University of California, Berkeley,
Space Astrophysics Group,
Berkeley, California

University of California, Irvine,
Information & Computer Science,
Irvine, California

University of California, Santa Barbara, *Santa Barbara, California*

University of Canterbury,
Christchurch, New Zealand

University College,
Cork, Ireland

University of Crete,
Institute of Computer Science,
Heraklio, Crete, Greece

University of Delaware,
Newark, Delaware

University of Exeter,
Computer Unit,
Exeter, Devon, England

University of Groningen,
Groningen, The Netherlands

University of Heidelberg,
Computing Center,
Heidelberg, Germany

University of Illinois at Chicago,
Computer Center,
Chicago, Illinois

Universität Koblenz–Landau,
Koblenz, Germany

University of Manitoba,
Winnipeg, Manitoba

University of Maryland,
Department of Computer Science,
College Park, Maryland

Università degli Studi di Trento,
Trento, Italy

University of Oslo,
Institute of Informatics,
Blindern, Oslo, Norway

University of Salford,
Salford, England

University of South Carolina,
Department of Mathematics,
Columbia, South Carolina

University of Southern California,
Information Sciences Institute,
Marina del Rey, California

University of Stockholm,
Department of Mathematics,
Stockholm, Sweden

University of Texas at Austin,
Austin, Texas

University of Washington,
Department of Computer Science,
Seattle, Washington

Uppsala University,
Uppsala, Sweden

Villanova University,
Villanova, Pennsylvania

Virginia Polytechnic Institute,
Interdisciplinary Center
for Applied Mathematics,
Blacksburg, Virginia

Vrije Universiteit,
Amsterdam, The Netherlands

Washington State University,
Pullman, Washington

Wolters Kluwer,
Dordrecht, The Netherlands

Yale University,
Department of Computer Science,
New Haven, Connecticut

Calendar

1993

Oct 4 – 8	CyrTUG meeting, Pereslavl′-Zalesskiĭ, near Pleshchevo Lake. For information, contact Irina Makhovaya (`cyrtug@mir.msk.su`).
Oct 7	TEX–Stammtisch at the Universität Bremen, Germany. For information, contact Martin Schröder (`115d@alf.zfn.uni-bremen.de`; telephone 0421/628813).
Oct 18 – 22	**TUG Course:** Beginning/Intermediate TEX Santa Barbara, California
Oct 18	TEX-Stammtisch in Bonn, Germany. For information, contact Herbert Framke (`Herbert_Framke@BN.MAUS.DE`; telephone 02241 400018).
Oct 19	TEX-Stammtisch in Duisburg, Germany. For information, contact Friedhelm Sowa (`tex@ze8.rz.uni-duesseldorf.de`; telephone 0211/311 3913).
Oct 27	TEX–Stammtisch, Hamburg, Germany. For information, contact Reinhard Zierke (`zierke@informatik.uni-hamburg.de`; telephone (040) 54715-295).

TUG Courses, Santa Barbara, California

Oct 25 – 29	Intensive LATEX
Nov 1 – 5	Advanced TEX and Macro Writing
Nov 8 – 9	Practical SGML and TEX

Nov 4	TEX–Stammtisch at the Universität Bremen, Germany. (For contact information, see Oct 7.)
Nov 10	**TUG Course:** SGML and TEX for Publishers, New York City
Nov 12	**TUG Course:** TEX for Publishers, Washington, DC
Nov 15	**TUGboat Volume 15, 1st regular issue:** Deadline for receipt of *technical* manuscripts.
Nov 15	TEX-Stammtisch in Bonn, Germany. (For contact information, see Oct 18.)
Nov 16	TEX-Stammtisch in Duisburg, Germany. (For contact information, see Oct 19.)
Nov 18	12th NTG Meeting, "(LA)TEX user environment", Oce, Den Bosch, Netherlands. For information, contact Gerard van Nes (`vannes@ecn.nl`).
Nov 23	**TUGboat Volume 14, 3rd regular issue:** Mailing date (tentative).
Nov 24	TEX–Stammtisch, Hamburg, Germany. (For contact information, see Oct 27.)
Dec 2	TEX–Stammtisch at the Universität Bremen, Germany. (For contact information, see Oct 7.)
Dec 13	**TUGboat Volume 15, 1st regular issue:** Deadline for receipt of news items, reports.
Dec 20	TEX-Stammtisch in Bonn, Germany. (For contact information, see Oct 18.)
Dec 22	TEX–Stammtisch, Hamburg, Germany. (For contact information, see Oct 27.)
Dec 21	TEX-Stammtisch in Duisburg, Germany. (For contact information, see Oct 19.)

1994

TUG Courses, Santa Barbara, California

Jan 31 – Feb 4	Intensive LATEX
Feb 7 – 11	Beginning/Intermediate TEX
Feb 14 – 18	Advanced TEX and Macro Writing
Feb 28 – Mar 2	Modifying LATEX Style Files

Status as of 15 September 1993

Feb 15 **TUGboat Volume 15, 2nd regular issue:** Deadline for receipt of *technical* manuscripts (tentative).

Mar 9 **TUGboat Volume 15, 1st regular issue:** Mailing date (tentative).

Mar 15 **TUGboat Volume 15, 2nd regular issue:** Deadline for receipt of news items, reports (tentative).

Apr 11–15 Four conferences, Darmstadt, Germany:

- EP94, Electronic Pubishing, Document Manipulation and Typography (for information, contact `ep94@gmd.de`);
- RIDT94, Raster Imaging and Digital Typography (for information, contact `ridt94@irisa.fr`);
- TEP94, Teaching Electronic Publishing (for information, contact `ltsdyson@reading.ac.uk`);
- PODP94, Principles of Document Processing (for information, contact `podp94@cs.umd.edu`). Deadline for submission of papers: 15 August 1993

May 23 **TUGboat Volume 15, 2nd regular issue:** Mailing date (tentative).

Jul 24–29 SIGGRAPH'94: 21st International ACM Conference on Computer Graphics and Interactive Techniques. Orlando, Florida (for information, contact `siggraph-94@siggraph.org`, telephone 312-321-6830).

Jul 31 – Aug 4 **TUG Annual Meeting**, Santa Barbara, California. For information, contact the TUG office.

For additional information on the events listed above, contact the TUG office (805-963-1338, fax: 805-963-8358, email: `tug@tug.org`) unless otherwise noted.

Index of Advertisers

366	American Mathematical Society
366	ArborText
Cover 3	Blue Sky Research
367	Ed Baker Technical Services
364, 365	Kinch Computer Company
360	Micro Programs, Inc.
360	TEX Users Group
368	Y&Y

TUG'94

Santa Barbara, California
31 July – 4 August

Call for Papers

The **15th Anniversary Meeting** of the TeX Users Group will be held in Santa Barbara, California from 31 July through 4 August 1994. Those wishing to present papers must have their titles and outlines submitted to the program committee at `tug94@tug.org` by **February 1, 1994**, completed papers by **May 20, 1994.**

Since TeX and METAFONT applications and interests are as varied as our users, we are encouraging papers over the entire range of related topics, with a particular **focus on innovation.** Let us take a fresh look at what we have and envision new areas of use.

FOR YOUR TeX TOOLBOX

CAPTURE
Capture graphics generated by application programs. Make LaserJet images compatible with TeX. Create pk files from pcl or pcx files. $135.00

texpic
Use texpic graphics package to integrate simple graphics—boxes, circles, ellipses, lines, arrows—into your TeX documents. $79.00

Voyager
TeX macros to produce viewgraphs—including bar charts—quickly and easily. They provide format, indentation, font, and spacing control. $25.00

FOR YOUR TeX BOOKSHELF

TeX BY EXAMPLE NEW!
Input and output are shown side-by-side. Quickly see how to obtain desired output. $19.95

TeX BY TOPIC NEW!
Learn to program complicated macros. . . . $29.25

TeX FOR THE IMPATIENT
Includes a complete description of TeX's control sequences. $29.25

TeX FOR THE BEGINNER NEW!
A carefully paced tutorial introduction. . . $29.25

BEGINNER'S BOOK OF TeX
A friendly introduction for beginners and aspiring "wizards." $29.95

MR Micro Programs Inc. 251 Jackson Ave. Syosset, NY 11791 (516) 921-1351

TEX USERS GROUP

Individual Membership Application

Name ____________________

Institutional affiliation, if any ____________________

Position ____________________

Address (business or home (circle one)) ____________________

____

____

City ____________________ Province/State ____________

Country ____________________ Postal Code ____________

Telephone ____________________ FAX ____________

Email address ____________________

I am also a member of the following other TEX organizations:

Specific applications or reasons for interest in TEX:

There are two types of TUG members: regular members, who pay annual dues of $60; and full-time student members, whose annual dues are $30. Students must include verification of student status with their applications.

Please indicate the type of membership for which you are applying:

☐ Regular at $60 ☐ Full-time student at $30

Amount enclosed for 1993 membership: $ ________

☐ Check/money order payable to TEX Users Group enclosed (*checks in US dollars must be drawn on a US bank; checks in other currencies are acceptable, drawn on an appropriate bank*)

☐ Bank transfer:

TEX Users Group, Account #1558-816,
Santa Barbara Bank and Trust, 20 East Carrillo Street,
Santa Barbara, CA 93101 USA

your bank ____________________

ref # ____________________

☐ Charge to MasterCard/VISA

Card # ____________________ Exp. date ________

Signature ____________________

Complete and return this form with payment to:

TEX Users Group
Membership Department
P. O. Box 869
Santa Barbara, CA 93102 USA

Telephone: (805) 963-1338
FAX: (805) 963-8358
Email: `tug@tug.org`

Membership is effective from January 1 to December 31 and includes subscriptions to *TUGboat, The Communications of the TEX Users Group* and the TUG newsletter, *TEX and TUG NEWS*. Members who join after January 1 will receive all issues published that calendar year.

For more information ...

Whether or not you join TUG now, feel free to return this form to request more information. Be sure to include your name and address in the spaces provided to the right.

Check all items you wish to receive below:

☐ Institutional membership information

☐ Course and meeting information

☐ Advertising rates

☐ Products/publications catalogue

☐ Public domain software catalogue

☐ More information on TEX

TEX USERS GROUP

Institutional Membership Application

Institution or Organization ______________________

____

Principal contact ______________________

Address ______________________

____

City ____________ Province/State ____________

Country ____________ Postal Code ____________

Daytime telephone ____________ FAX ____________

Email address ______________________

Complete and return this form with payment to:

TEX Users Group
Membership Department
P. O. Box 21041
Santa Barbara, CA 93121-1041
USA

Membership is effective from January 1 to December 31. Members who join after January 1 will receive all issues of *TUGboat* and *TEX and TUG NEWS* published that calendar year.

Each Institutional Membership entitles the institution to:

- designate a number of individuals to have full status as TUG individual members;
- take advantage of reduced rates for TUG meetings and courses for *all* staff members;
- be acknowledged in every issue of *TUGboat* published during the membership year.

Educational institutions receive a $100 discount in the membership fee. The three basic categories of Institutional Membership each include a certain number of individual memberships. Additional individual memberships may be obtained at the rates indicated. Fees are as follows:

Category	Rate (educ. / non-educ.)	Add'l mem.
A (includes 7 memberships)	$ 540 / $ 640	$50 ea.
B (includes 12 memberships)	$ 815 / $ 915	$50 ea.
C (includes 30 memberships)	$1710 / $1810	$40 ea.

Please indicate the type of membership for which you are applying:

Category _____ + _____ additional individual memberships

Amount enclosed for 1993 membership: $ ________

☐ Check/money order payable to TEX Users Group enclosed (*payment in US dollars must be drawn on a US bank; payment in other currencies is acceptable, drawn on an appropriate bank*)

☐ Bank transfer: your bank ______________________

ref # ______________________

TEX Users Group, Account #1558-816,
Santa Barbara Bank and Trust, 20 East Carrillo Street,
Santa Barbara, CA 93101 USA

☐ Charge to MasterCard/VISA

Card # ______________________ Exp. date ________

Signature ______________________

For more information ...

Correspondence

TEX Users Group
P. O. Box 869
Santa Barbara, CA 93102
USA

Telephone: (805) 963-1338
FAX: (805) 963-8358
Email: `tug@tug.org`

Whether or not you join TUG now, feel free to return this form to request more information.

Check all items you wish to receive below:

☐ Course and meeting information
☐ Advertising rates
☐ Products/publications catalogue
☐ Public domain software catalogue
☐ More information on TEX

Please attach a list of individuals whom you wish to designate as TUG individual members. Minimally, we require names and addresses so that TUG publications may be sent directly to these individuals, but we would also appreciate receiving the supplemental information regarding phone numbers, email addresses, and TEX interests as requested on the TUG Individual Membership Application form. For this purpose, the latter application form may be photocopied and mailed with this form.

TEX Consulting & Production Services

North America

Abrahams, Paul

214 River Road, Deerfield, MA 01342; (413) 774-5500

Development of TEX macros and macro packages. Short courses in TEX. Editing assistance for authors of technical articles, particularly those whose native language is not English My background includes programming, computer science, mathematics, and authorship of *TEX for the Impatient.*

American Mathematical Society

P. O. Box 6248, Providence, RI 02940; (401) 455-4060

Typesetting from DVI files on an Autologic APS Micro-5 or an Agfa Compugraphic 9600 (PostScript). Times Roman and Computer Modern fonts. Composition services for mathematical and technical books and journal production.

Anagnostopoulos, Paul C.

433 Rutland Street, Carlisle, MA 01741; (508) 371-2316

Composition and typesetting of high-quality books and technical documents. Production using Computer Modern or any available PostScript fonts. Assistance with book design. I am a computer consultant with a Computer Science education.

ArborText, Inc.

1000 Victors Way, Suite 400, Ann Arbor, MI 48108; (313) 996-3566

TEX installation and applications support. TEX-related software products.

Archetype Publishing, Inc., Lori McWilliam Pickert

P. O. Box 6567, Champaign, IL 61821; (217) 359-8178

Experienced in producing and editing technical journals with TEX; complete book production from manuscript to camera-ready copy; TEX macro writing including complete macro packages; consulting.

The Bartlett Press, Inc., Frederick H. Bartlett

Harrison Towers, 6F, 575 Easton Avenue, Somerset, NJ 08873; (201) 745-9412

Vast experience: 100+ macro packages, over 30,000 pages published with our macros; over a decade's experience in all facets of publishing, both TEX and non-TEX; all services from copyediting and design to final mechanicals.

Cowan, Dr. Ray F.

141 Del Medio Ave. #134, Mountain View, CA 94040; (415) 949-4911

Ten Years of TEX and Related Software Consulting, Books, Documentation, Journals, and Newsletters. TEX & LATEX macropackages, graphics; PostScript language applications; device drivers; fonts; systems.

Electronic Technical Publishing Services Co.

2906 Northeast Glisan Street, Portland, Oregon 97232-3295;
(503) 234-5522; FAX: (503) 234-5604

Total concept services include editorial, design, illustration, project management, composition and prepress. Our years of experience with TEX and other electronic tools have brought us the expertise to work effectively with publishers, editors, and authors. ETP supports the efforts of the TEX Users Group and the world-wide TEX community in the advancement of superior technical communications.

NAR Associates

817 Holly Drive E. Rt. 10, Annapolis, MD 21401; (410) 757-5724

Extensive long term experience in TEX book publishing with major publishers, working with authors or publishers to turn electronic copy into attractive books. We offer complete free lance production services, including design, copy editing, art sizing and layout, typesetting and repro production. We specialize in engineering, science, computers, computer graphics, aviation and medicine.

Ogawa, Arthur

1101 San Antonio Road, Suite 413, Mountain View, CA 94043-1002;
(415) 691-1126;
`ogawa@applelink.apple.com`.

Specialist in fine typography, LATEX book production systems, database publishing, and SGML. Programming services in TEX, LATEX, PostScript, SGML, DTDs, and general applications. Instruction in TEX, LATEX, and SGML. Custom fonts.

Pronk&Associates Inc.

1129 Leslie Street, Don Mills, Ontario, Canada M3C 2K5;
(416) 441-3760; Fax: (416) 441-9991

Complete design and production service. One, two and four-color books. Combine text, art and photography, then output directly to imposed film. Servicing the publishing community for ten years.

Quixote Digital Typography, Don Hosek

349 Springfield, #24, Claremont, CA 91711; (714) 621-1291

Complete line of TEX, LATEX, and METAFONT services including custom LATEX style files, complete book production from manuscript to camera-ready copy; custom font and logo design; installation of customized TEX environments; phone consulting service; database applications and more. Call for a free estimate.

Richert, Norman

1614 Loch Lake Drive, El Lago, TX 77586; (713) 326-2583

TEX macro consulting.

TEXnology, Inc., Amy Hendrickson

57 Longwood Ave., Brookline, MA 02146; (617) 738-8029

TEX macro writing (author of MacroTEX); custom macros to meet publisher's or designer's specifications; instruction.

Type 2000

16 Madrona Avenue, Mill Valley, CA 94941;
(415) 388-8873; FAX (415) 388-8865

$2.50 per page for 2000 DPI TEX camera ready output! We have a three year history of providing high quality and fast turnaround to dozens of publishers, journals, authors and consultants who use TEX. Computer Modern, Bitstream and METAFONT fonts available. We accept DVI files only and output on RC paper. $2.25 per page for 100+ pages, $2.00 per page for 500+ pages.

Outside North America

TypoTEX Ltd.

Electronical Publishing, Battyány u. 14. Budapest, Hungary H-1015; (036) 11152 337

Editing and typesetting technical journals and books with TEX from manuscript to camera ready copy. Macro writing, font designing, TEX consulting and teaching.

Information about these services can be obtained from:

TEX Users Group
P. O. Box 869
Santa Barbara, CA 93102-0869
Phone: (805) 963-1388
Fax: (805) 963-8538

AP-TEX Fonts

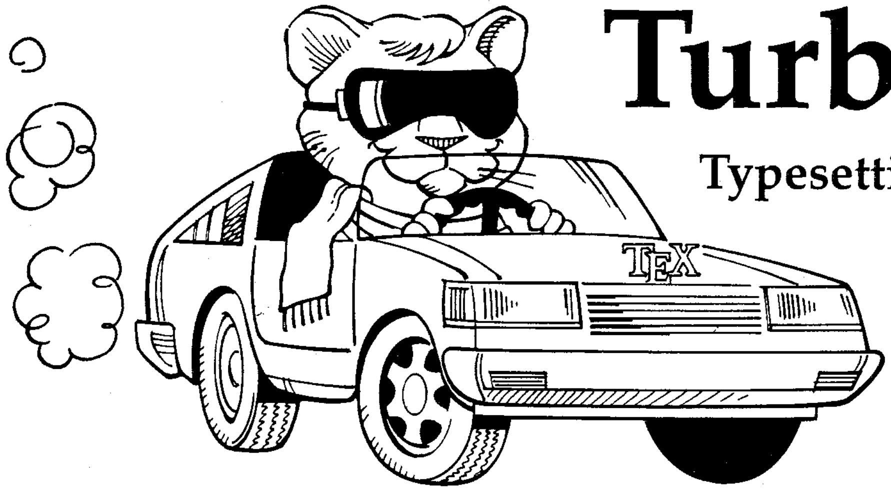
TEX-compatible Bit-Mapped Fonts
Identical to
Adobe PostScript Typefaces

If you are hungry for new TEX fonts, here is a feast guaranteed to satisfy the biggest appetite! The AP-TEX fonts serve you a banquet of gourmet delights: 438 fonts covering 18 sizes of 35 styles, at a total price of $200. The AP-TEX fonts consist of `PK` and `TFM` files which are exact TEX-compatible equivalents (including "hinted" pixels) to the popular PostScript name-brand fonts shown at the right. Since they are directly compatible with any standard TEX implementation (including kerning and ligatures), you don't have to be a TEX expert to install or use them.

When ordering, specify resolution of 300 dpi (for laser printers), 180 dpi (for 24-pin dot matrix printers), or 118 dpi (for previewers). Each set is on ten 360 KB 5-1/4" PC floppy disks. The $200 price applies to the first set you order; order additional sets at other resolutions for $60 each. A 30-page user's guide fully explains how to install and use the fonts. Sizes included are 5, 6, 7, 8, 9, 10, 11, 12, 14.4, 17.3, 20.7, and 24.9 points; headline styles (equivalent to Times Roman, Helvetica, and Palatino, all in bold) also include sizes 29.9, 35.8, 43.0, 51.6, 61.9, and 74.3 points.

The Kinch Computer Company
PUBLISHERS OF TURBOTEX
501 South Meadow Street
Ithaca, New York 14850
Telephone (607) 273-0222
FAX (607) 273-0484

Avant Garde	Bold
Avant Garde	Bold Oblique
Avant Garde	Demibold
Avant Garde	Demibold Oblique
Bookman	Light
Bookman	Light Italic
Bookman	Demibold
Bookman	Demibold Italic
Courier	
Courier	Oblique
Courier	Bold
Courier	Bold Oblique
Helvetica	
Helvetica	Oblique
Helvetica	Bold
Helvetica	Bold Oblique
Helvetica Narrow	
Helvetica Narrow	Oblique
Helvetica Narrow	Bold
Helvetica Narrow	Bold Oblique
Schoolbook	New Century Roman
Schoolbook	New Century Italic
Schoolbook	New Century Bold
Schoolbook	New Century Bold Italic
Palatino	Roman
Palatino	Italic
Palatino	Bold
Palatino	Bold Italic
Times	Roman
Times	Italic
Times	Bold
Times	Bold Italic
Zapf Chancery	Medium Italic
Symbol ΔΦΓϑΛΠΘ	
Zapf Dingbats ✄☞❑	



TurboTEX

Typesetting Software

Executables $150
With Source $300

MICROSOFT® WINDOWS™ COMPATIBLE

THE MOST VERSATILE TEX ever published is breaking new ground in the powerful and convenient graphical environment of Microsoft Windows: TurboTEX Release 3.1E. TurboTEX runs on all the most popular operating systems (Windows, MS-DOS, OS/2, and UNIX) and provides the latest TEX 3.14 and METAFONT 2.7 standards and certifications: preloaded plain TEX, LATEX, AMS-TEX and AMS-LATEX, previewers for PC's and X-servers, METAFONT, Computer Modern and LATEX fonts, and printer drivers for HP LaserJet and DeskJet, PostScript, and Epson LQ and FX dot-matrix printers.

■ **Best-selling Value:** TurboTEX sets the world standard for power and value among TEX implementations: one price buys a complete, commercially-hardened typesetting system. *Computer* magazine recommended it as "the version of TEX to have," *IEEE Software* called it "industrial strength," and thousands of satisfied users around the globe agree.

TurboTEX gets you started quickly, installing itself automatically under MS-DOS or Microsoft Windows, and compiling itself automatically under UNIX. The 90-page User's Guide includes generous examples and a full index, and leads you step-by-step through installing and using TEX and METAFONT.

■ **Classic TEX for Windows.** Even if you have never used Windows on your PC, the speed and power of TurboTEX will convince you of the benefits. While the TEX command-line options and *TEXbook* interaction work the same, you also can control TEX using friendly icons, menus, and dialog boxes. Windows protected mode frees you from MS-DOS limitations like DOS extenders, overlay swapping, and scarce memory. You can run long TEX formatting or printing jobs in the background while using other programs in the foreground.

■ **MS-DOS Power, Too:** TurboTEX still includes the plain MS-DOS programs. Virtual memory simulation provides the same sized TEX that runs on multi-megabyte mainframes, with capacity for large documents, complicated formats, and demanding macro packages.

■ **Source Code:** The portable C source to TurboTEX consists of over 100,000 lines of generously commented TEX, TurboTEX, METAFONT, previewer, and printer driver source code, including: our WEB system in C; PASCHAL, our proprietary Pascal-to-C translator; Windows interface; and preloading, virtual memory, and graphics code, all meeting C portability standards like ANSI and K&R.

■ **Availability & Requirements:** TurboTEX executables for IBM PC's include the User's Guide and require 640K, hard disk, and MS-DOS 3.0 or later. Windows versions run on Microsoft Windows 3.0 or 3.1. Order source code (includes Programmer's Guide) for other machines. On the PC, source compiles with Microsoft C, Watcom C 8.0, or Borland C++ 2.0; other operating systems need a 32-bit C compiler supporting UNIX standard I/O. Specify 5-1/4" or 3-1/2" PC-format floppy disks.

■ **Upgrade at Low Cost.** If you have TurboTEX Release 3.0, upgrade to the latest version for just $40 (executables) or $80 (including source). Or, get either applicable upgrade free when you buy the AP-TEX fonts (see facing page) for $200!

■ **No-risk trial offer:** Examine the documentation and run the PC TurboTEX for 10 days. If you are not satisfied, return it for a 100% refund or credit. (Offer applies to PC executables only.)

■ **Free Buyer's Guide:** Ask for the free, 70-page Buyer's Guide for details on TurboTEX and dozens of TEX-related products: previewers, TEX-to-FAX and TEX-to-Ventura/Pagemaker translators, optional fonts, graphics editors, public domain TEX accessory software, books and reports.

Ordering TurboTEX

Ordering TurboTEX is easy and delivery is fast, by phone, FAX, or mail. Terms: Check with order (free media and ground shipping in US), VISA, Mastercard (free media, shipping extra); Net 30 to well-rated firms and public agencies (shipping and media extra). Discounts available for quantities or resale. International orders gladly expedited via Air or Express Mail.

The Kinch Computer Company
PUBLISHERS OF TURBOTEX
501 South Meadow Street
Ithaca, New York 14850 USA
Telephone (607) 273-0222
FAX (607) 273-0484

TEX Publishing Services

From the Basic:
The American Mathematical Society offers you two basic, low cost TEX publishing services.

- You provide a DVI file and we will produce typeset pages using an Autologic APS Micro-5 phototypesetter. $5 per page for the first 100 pages; $2.50 per page for additional pages.
- You provide a PostScript output file and we will provide typeset pages using an Agfa/ Compugraphic 9600 imagesetter. $7 per page for the first 100 pages; $3.50 per page for additional pages.

There is a $30 minimum charge for either service. Quick turnaround is also provided... a manuscript up to 500 pages can be back in your hands in one week or less.

To the Complex:
As a full-service TEX publisher, you can look to the American Mathematical Society as a single source for any or all your publishing needs.

Macro-Writing	TEX Problem Solving	Non-CM Fonts	Keyboarding
Art and Pasteup	Camera Work	Printing and Binding	Distribution

For more information or to schedule a job, please contact Regina Girouard, American Mathematical Society, P. O. Box 6248, Providence, RI 02940, or call 401-455-4060.

NEW! NEW!

ARBORTEXT INC.

- Silicon Graphics Iris or Indigo
- Solaris 2.1
- DVILASER/HP3
- Motif and OPEN LOOK Preview

Complete TEX packages
Ready to use, fully documented and supported.

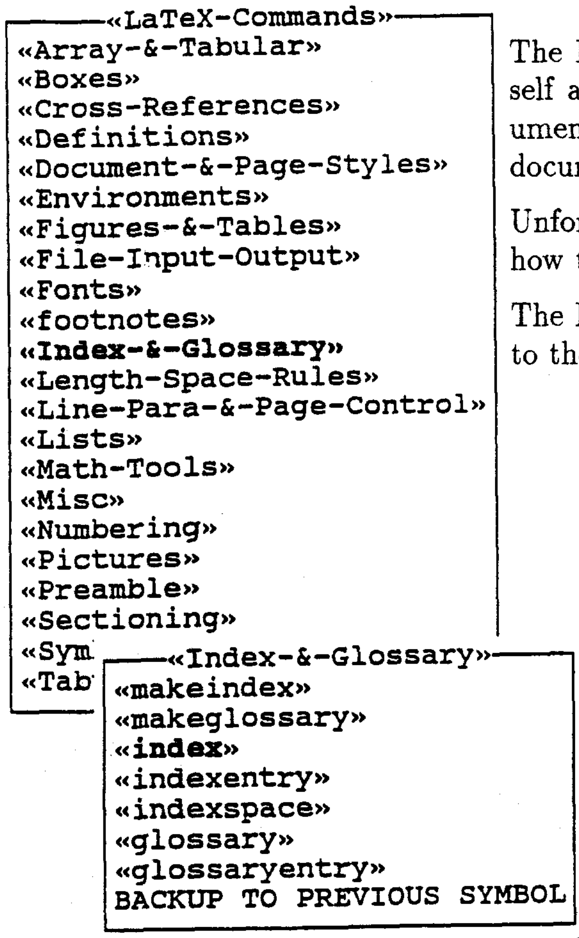
Also Available For: Sun-4 (SPARC), IBM RS/6000, DEC/RISC-Ultrix, HP 9000, and IBM PC's

Call us for more information on our exciting new products!

1000 Victors Way ▲ Suite 400 ▲ Ann Arbor, MI 48108 ▲ (313) 996–3566 ▲ FAX (313) 996–3573

LACOD

A LaTeX Source Code Development System



```
——————«LaTeX-Commands»——————
«Array-&-Tabular»
«Boxes»
«Cross-References»
«Definitions»
«Document-&-Page-Styles»
«Environments»
«Figures-&-Tables»
«File-Input-Output»
«Fonts»
«footnotes»
«Index-&-Glossary»
«Length-Space-Rules»
«Line-Para-&-Page-Control»
«Lists»
«Math-Tools»
«Misc»
«Numbering»
«Pictures»
«Preamble»
«Sectioning»
«Sym
«Tab
```

```
———«Index-&-Glossary»———
«makeindex»
«makeglossary»
«index»
«indexentry»
«indexspace»
«glossary»
«glossaryentry»
BACKUP TO PREVIOUS SYMBOL
```

The LaTeX document preparation system has proven itself as a tremendous system for creating technical documents. It is a feature rich system that can produce documents to the highest standards of typography.

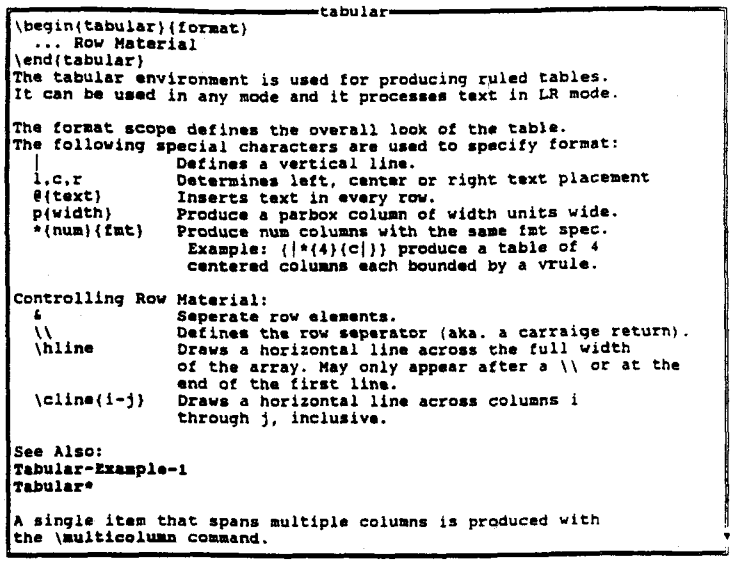
Unfortunately, even if you use it everyday, remembering how to use all those features is next to impossible.

The LACOD system brings a new, more user friendly, face to the creation of LaTeX documents.

A Pop-Up command selector contains *all* LaTeX commands. You simply select a command and LACOD will prompt you for all options and fill-in information. LACOD then inserts the syntactically correct command, or environment, into your document.

When you need help, or examples for reference, place the cursor under *any* LaTeX command and request Hyper-Help. A Pop-Up screen will display proper command syntax and a complete example of its use. Many screens are cross-referenced by Hyper-Links to related commands and examples.

- **Pop-Up LaTeX Command Menu**
- **Pop-Up LaTeX Hyper-Help**
- **OS/2 and DOS Compatible**
- **10 Editing Windows**
- **Versatile Macro Capability**
- **Unlimited Undo**
- **Regular Expression Search**
- **Journaling w/playback**



```
——————————————————tabular——————————————————
\begin{tabular}{format}
  ... Row Material
\end{tabular}
The tabular environment is used for producing ruled tables.
It can be used in any mode and it processes text in LR mode.

The format scope defines the overall look of the table.
The following special characters are used to specify format:
  |              Defines a vertical line.
  l,c,r          Determines left, center or right text placement
  @{text}        Inserts text in every row.
  p{width}       Produce a parbox column of width units wide.
  *{num}{fmt}    Produce num columns with the same fmt spec.
                  Example: {|*{4}{c|}} produce a table of 4
                  centered columns each bounded by a vrule.

Controlling Row Material:
  &              Seperate row elements.
  \\             Defines the row seperator (aka. a carraige return).
  \hline         Draws a horizontal line across the full width
                 of the array. May only appear after a \\ or at the
                 end of the first line.
  \cline{i-j}    Draws a horizontal line across columns i
                 through j, inclusive.

See Also:
Tabular-Example-1
Tabular*

A single item that spans multiple columns is produced with
the \multicolumn command.
```

$65.00

Includes 2nd Day Priority Mail Shipping for U.S. customers. Shipping not included for international orders. However, international orders gladly expedited via Air or Express Mail.

30–Day Money Back Guarantee

EBTS	PO BOX 642L	Norfolk, MA	02056
TEL: 508-528-7728	FAX: 508-520-3272	email: ejb@world.std.com	

Lucida Bright + Lucida New Math

The Lucida Bright + Lucida New Math

font set is the first alternative to Computer Modern complete with math fonts in ATM compatible Adobe Type 1 format. Lucida Bright + Lucida New Math will give your papers and books a bright new look!

The Lucida New Math

fonts (Lucida New Math Italic, Lucida New Math Symbol, Lucida New Math Extension and Lucida New Math Arrows) include the mathematical signs and symbols most used in mathematical and technical composition, including italic Greek capitals and lower case. The Lucida New Math fonts contain the math characters that are standard in the TEX math composition software, which, in its various forms, is one of the most popular mathematical composition packages used worldwide. In addition to the standard Computer Modern math font character sets, Lucida New Math fonts also include the characters in the American Mathematical Society ($\mathcal{AMS}$) symbol fonts.

Switching to Lucida Bright + Lucida New Math is as easy as adding an input statement to the head of your TEX source file. Aside from four styles of each of the expected seriffed, sans serif, and fixed width text faces, the font set also contains Lucida Blackletter, Lucida Calligraphy and Lucida Handwriting.

The Lucida Bright + Lucida New Math

font set is available in fully hinted ATM compatible Adobe Type 1 format for Macintosh, IBM PC compatibles, as well as Unix/NeXT.

We also carry the other font sets commonly used with TEX in fully hinted ATM compatible Adobe Type 1 format, but we are most excited about the new Lucida Bright + Lucida New Math fonts. The finest tools for working with scalable outline fonts in TEX are DVIPSONE and DVIWindo (on IBM PC compatibles).

Y&Y, Inc.
106 Indian Hill
Carlisle, MA 01741
(800) 742-4059 (508) 371-3286 (voice) (508) 371-2004 (fax)

Interactive TEX ✓

WYSIWYG TEX ✓

User-friendly TEX ✓

Textures ✓ It's not like any other TEX system.[1]

When Apple introduced the Macintosh and its graphic interface, we embarked on a long-term project of research toward a TEX system "for the rest of us," a TEX system that would be humanely interactive, and visibly responsive; an integrated TEX *system*, designed to be radically easy for those new to TEX, and engineered to be the best for expert TEX users. The research continues; the product is Textures.

Textures is something of a Copernican revolution in TEX interface. The paradigm shifts from the usual TEX "input–process–output–repeat" mode, to a wider frame wherein the TEX language describes a dynamic document that is continuously, quietly "realized" as you write it, with no process steps whatsoever.

This change in perspective must be experienced to be fully grasped. As you would expect, Textures is good for beginners. You might be surprised to know that it's also good for experienced TEX users and especially good for TEX programmers.[2] It's not a "front-end" or an imitation, it's a full-scale live TEX processor that's actually easy to use.

There's much more to Textures than a new perspective on TEX, of course; too much to list here but we'll mention custom menus, Computer Modern PostScript fonts, illustrations, inter-application communication, automatic installation, genuine support, ...

We don't claim perfection; nor do we believe in exaggerated marketing, odd as this may seem; and we do understand our finite abilities to provide all that one might wish. But we also guarantee that you will be satisfied with Textures and with the service you receive from Blue Sky Research, or we will refund the (very finite) price you pay.

[1]
On the other hand, Textures is *exactly* like every other TEX system. Its TEX engine is strictly standard and up-to-date; it runs LATEX, AMS-TEX, and all standard TEX macros without change. But even here it's not ordinary, with hand-tuned assembler code for maximum performance, and a transparent memory model that you can't fill until you run out of disk.

[2]
If you are a serious TEX user on another platform, it can be worth getting a Mac just to run Textures.

For all Macintosh processors and printers minimum 2.5MB memory and 5MB disk

Blue Sky Research
534 SW Third Avenue
Portland, Oregon 97204
USA

800 622 8398
503 222 9571
facsimile 503 222 1643
sales@bluesky.com

TUGboat

Volume 14, Number 3 / October 1993
1993 Annual Meeting Proceedings