TUGboat

Volume 20, Number 2 / June 1999

Just as the terms *bureaucracy* and *bureaucrat* long ago came to suggest narrow outlook, lack of humanity, and otherwise vague reprobation, similar connotations have been attached to computer technology and personnel, although less frequently as a laity begins to take responsibilities from the priesthood.

James R. Beniger
The Control Revolution: Technical and Economic Origins of the Information Society (1986)

TUGboat

COMMUNICATIONS OF THE TEX USERS GROUP
EDITOR BARBARA BEETON

VOLUME 20, NUMBER 2 • JUNE 1999
PORTLAND • OREGON • U.S.A.

General Delivery

From the President

Mimi L. Jett

Greetings, fellow TUG members!

Happy 20th! Two decades of TUG, and our future is brighter than ever. With the widespread acceptance of structured markup languages, and a worldwide movement toward open standards, TEX is being recognized as an early front-runner, "way before its time" some say. The community of TEX users has been sharing ideas and technology since the beginning, and the result has benefited people in dozens of countries for thousands of projects. As the world of print publishing evolved into digital delivery, the TEX community has responded by creating programs that work with and around other technologies. Now that MathML is becoming the standard for mathematical markup, we see TEX as the math input language as well as the typesetting engine to back up XML. How convenient that scores of thousands of people already use it! It has been an interesting 20 years, full of technological miracles. May our next 20 be so fruitful.

With the first half of the year already behind us, it is time to look at our progress against the goals we set down at the end of last year. One shining star is our office, where the staff has finally tamed the database and established procedures to control membership records. There are occasional glitches, an unavoidable fact in most systems. Being very close to the situation (the office is merely 3 miles from my home), I can report that TUG is running smoothly, providing better service to the members than we have in a very long time. All this is accomplished with a staff of 2 half-time workers, and an occasional intern from Portland State University. Another group of stars, truly the solar system of our little community, is the board of directors. We welcome Stephanie Hogue and Cheryl Ponchin to the board! It is a huge advantage to TUG that the entire board has returned to serve another term. Learning to work together, simply understanding each other's strengths and weaknesses, takes a long time. Compound the distance between us — from Australia to the Czech Republic — with the diversity of our life's experiences, and you see why it takes the first years to discover how to work together not only efficiently, but also enjoyably. We may not always succeed, but we continue to try. There are so many hours spent on committees, projects, communications, and administration that are not always apparent, yet this volunteerism fuels the advances we share and improvements in the benefits we offer. The executive committee has worked very closely to oversee the works, meeting monthly with the business committee and office staff to set priorities and review progress. Our financial condition is often discussed at length, as we review reports and statements together via teleconferencing. Such scrutiny contributes to the understanding and accuracy of our financial reports, which will be presented during the annual general meeting at TUG '99 in Vancouver, BC. Absolute control of our financial affairs was a goal not very long ago. We have achieved it nicely with the help of our treasurer, Don DeLand, and the diligence of our office staff. Thanks to all!

This has been a year of growth and change for me personally. After more than 20 years of entrepreneurism, I am happy to be part of one of the biggest and greatest companies on earth, IBM. Ironically, it was a presentation at EuroTEX in St. Malo that changed my outlook on Web publishing, and then my life. Bob Sutor and Angel Diaz gave a such a compelling demonstration of live math from LATEX code, it was impossible not to tell the world what they had accomplished. After several months of cheering from the sidelines, they invited me to join the team. I have the best job possible, evangelizing a very cool product. In addition to working hard on XML technologies, and contributing to many W3C committees, IBM also supports TUG in a big way. All my lucky stars lined up for this wonderful outcome, and I am very thankful.

In this issue you will find a set of 3 CDs containing the latest CTAN archive. This distribution is a benefit of membership, we hope you find it useful and valuable. Please see page 127 for an article describing the contents and installation of the software. Please let us know what you think. Send your comments to `tug-pubs@tug.org`.

Don't forget the upcoming annual meeting and conference TUG '99 to be held in Vancouver in British Columbia August 15–19. The quality of papers and panel discussions, in addition to the outstanding venue, should contribute to one of the best conferences we have had. The University of British Columbia is a beautiful setting, and Vancouver is one of the finest cities in North America. If you are not able to make the trek to Canada, plan for the EuroTEX conference in Heidelberg, "Paperless TEX", September 20–23, which also has a strong program and promises a wonderful time. Both will be great!

Finally, a note of thanks to all of our members, past and present. Over the years we have had thousands of people support our work for the TEX community. By joining TUG and contributing to our publications and projects, you are contributing to a worldwide consortium and expanding knowledge base dedicated to mathematics. Thank you, members!

◇ Mimi L. Jett
IBM
T. J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598
jett@us.ibm.com

Editorial Comments

Barbara Beeton

Remembering

One of the great pleasures of TUG is meeting so many interesting people. This is balanced by sorrows, when some of these people are taken from us.

Earlier this year, two deaths occurred in the TEX family — Roswitha Graham and Norman Naugle. Later on in these pages, you will find a brief remembrance of each of them, written by a colleague. But let me do a little remembering here.

I met Roswitha in 1989, when she attended the annual meeting at the invitation of Bart Childs, then president of TUG, along with the heads of several other European TEX groups, to share their knowledge of the many users of TEX in their groups who were not members of TUG. These individuals sat on the TUG board for several years as Special Directors, and we came to know one another in that context. Roswitha was always concerned about what was best for the users and the organization; she took this responsibility very seriously, but with great charm and dignity. In the spring of 1992, after a standards meeting in Copenhagen, my husband joined me for a trip through parts of Scandinavia. While in Stockholm, we visited with Roswitha, both in town and on the Grahams' island in the archipelago outside the harbor. During that visit, she showed us where Don Knuth had found a real web — constructed by a diligent spider — and she and I made arrangements to transcribe the tape of Don's Q & A session following the presentation of his honorary doctorate from the Royal Institute of Technology (see *TUGboat* **13**(4), pp. 419 *ff*). I remember Roswitha as a gracious and capable person, welcoming and caring.

Norman Naugle burst upon the TEX scene much earlier; his name first appears in a membership list in 1982, and I met him that year at the annual meeting at Stanford. Outgoing, exuberant, all the adjectives one can think of to describe a Texas native (or TEXas as it would later appear in the program of the 1990 annual meeting in College Station). Norm didn't want anything to do with the TUG bureaucracy, but he surely spread the word about TEX throughout Texas A&M — among other things, he is responsible for Bart Childs becoming president of TUG, and he nurtured a number of bright and enthusiastic students, among them Tom Rokicki, whom he mentored as an undergraduate. When I asked Tom if he had any thoughts in memory of Norm, he sent me this message.

> Norman Naugle was a hero. He was my mentor at Texas A&M and he always seemed to have a job or a programming task that challenged and rewarded me. But more than anything else he was a great friend. He charmed my mother by bragging about her son, invited me to dinner when I couldn't make it home over spring break, and generously shared his computing toys. It is my great fortune to have known him, and I will miss him greatly.

It has been a privilege to know both Roswitha and Norm, and I will miss them both.

New home for the UK TUG FAQ

Not long after the UK TUG FAQ was first published on paper, in the UK TUG Journal Baskerville (vol. 4, no. 6, December 1994), the group established a "temporary" Web address for interactive access to the FAQ. The School of Cognitive and Computing Sciences of the University of Sussex at Brighton was host to this "temporary" service for about four years, a service for which all TEX users are grateful.

Now, at last, a "final" home has been announced by the UK TUG, established in association with the CTAN node at the University of Cambridge Computer Laborarory: `http://www.tex.ac.uk/cgi-bin/texfaq2html?introduction=yes`

The sources, and readily-printable copies, of the FAQ remain on CTAN in directory `usergrps/uktug/faq`.

The interactive FAQ still offers the same facilities as it always has, but there are plans to develop new facilities to further enhance its utility.

The FAQ is under constant development, and in particular a new printed version is in preparation. The UK TEX Users' Group would very much welcome contributions at this time.

Comments, suggestions and error reports concerning the FAQ should be addressed to the current maintainer, via `uktug-faq@tex.ac.uk`.

Thanks to Robin Fairbairns for this report.

TUGboat authors' rights

In the last issue, we stated our policy about making *TUGboat* available in electronic form, via the *TUGboat* web pages. Although it was stated that authors retain the copyright to their own articles, we neglected to mention that TUG has no objection to authors posting this material on their own Web pages, or including it with packages on CTAN. Once an article has appeared in *TUGboat*, the author is welcome to include the reference in a footnote; if an author requests it, we will return the file as published for the author's use.

In fact, we encourage authors to keep their articles updated, if information in them is subject to change — what will appear on the *TUGboat* pages is just what appeared in print, except for application of errata and corrigenda, if any such are called to our attention.

Home site for ConTEXt

In the last issue, it was announced that the ConTEXt system, by Hans Hagen, was available at CTAN. Hans has reminded me that although the macros and maybe a few manuals are on CTAN, the main site is actually `www.pragma-ade.nl`. Here one can find about 50 Mb of macros, documentation, examples, and more; this is expected to double as Hans finds the time to sort things out. Since CTAN has limited disk space, only the most important pieces will be found there.

Credit where credit is due

The article on Father Truchet in the last issue of *TUGboat* was written by both Jacques André and Denis Girou. Unfortunately, Denis' name was omitted from the table of contents. Apologies!

This omission has been rectified in the online version of the contents and in Nelson Beebe's *TUGboat* bibliography.

The growing Russian TEX library

We have been informed that *The LATEX Companion* has joined the collection of TEX-related books now available in Russian. If you are interested in this edition, by Mir Publishers, please get in touch with Irina Makhovaya, the Executive Director of CyrTUG, at `irina@mir.msk.su`.

A new feature: Cartoons by Roy Preston

For about a year, visitors to the Typo-L web site have been amused by topical cartoons by Roy Preston. Topics have been wide-ranging: type identification and usage, commentary on particular types or typographers, questions about copyright protection, even a few *ad hominem* items inspired by various list correspondents. Many of the topics are (or should be) familiar to *TUGboat* readers, so I asked Roy for permission to publish some of his cartoons — and he has generously granted it.

Roy is a semi-retired former illustrator/graphic designer/art director/creative director with 25 years of experience in advertising. He lives with his family of three cats in Hardy's Dorset, England, and spends his waking time painting, designing fonts, and indulging in discussions on Typo-L.

You can see some of Roy's cartoons (and a lot of other interesting typo-related material) on the Typo-L web site, at `http://www.ids.co.uk/preston/typo/`. If you'd like to subscribe to the mailing list, instructions are on the web site. Typo-L was started as a TEX-related list (the "listmom" is our good friend Peter Flynn), though it has since been adopted by "mainline" type mavens; however, there are still a number of TEXies in the ranks.

So watch for the cartoons. I think we're in for a treat.

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

Norman W. Naugle
A Rememberance

Bart Childs

Norman Wakefield Naugle died on Friday, July 1, 1999. He was 68 years old and had been suffering from the dreaded Alzheimer's. Still, to the end he was pretty much the Norman that we knew and recall so fondly.

A few years ago he married Esta, a friend from high school times. She was a great partner for him and with his son Ross and daughter Nancy gave him the care needed toward the end. They finally had him admitted to a specialized care facility and he died two weeks later. It is such an unfair disease.

Norman was from Saginaw, Texas, which is now nearly swallowed from the northward expansion of Ft. Worth. He finished high school and immediately pursued his B.S. at Texas A&M. He told me that he wanted to be an electrical engineer, but that he quit because they required him to take too many power courses and discouraged his studying electronics.

In those days this was a small school, all male, and all military. He identified with his Corps of Cadets unit, the Signal Corps. He remained active in the alumni affiliations with that until just recently. Esta attended some of their functions with him.

Norman was last in my office in early May. I saw him on his bicycle later in the month. He knew why he wanted to be there, to talk about helping people understand and use TEX/LATEX. It hurt because he could not find or remember people's names or the city they were in. If he was trying to indicate Don Knuth, Tom Rokicki, or someone in Austin, he would gesture to the west.

Norman introduced me to TEX. I was supporting a basic word processor in our department. I made a large number of extensions to it including going into graphics mode and beating out an integral sign by repeated use of the period. We saw each other at the university dairy bar one day and I told him I wanted to show him this. He responded that he would like to see it but wanted to show me something when I finished. He brought the original TEX and METAFONT book from Digital Press. I do not think I ever touched that word processor again.

Dave Kellerman approached Norman about being President of TUG. Norman deflected that toward me. That opportunity to serve has certainly been one of the highlights of my professional career and I will be forever grateful. Norman often spent his own money to get release tapes of the TEX systems during our development days. He was that kind of giving and unselfish guy.

At his memorial service, Carl Pearcy told about one of their colleagues asking Norman to turn in his dissertation to the library, where it was scrutinized with great care. It came back with a large number of necessary changes to be accepted. Norman quietly sat down and retyped the colleague's dissertation. Carl also pointed out that Norman finished his Ph.D. while working at NASA on the Lunar Landing. He was the person responsible for the mapping of the lunar surface, and he did it!

Norman loved Texas A&M and most things about it. He would stop and pick up discarded drink containers, newspapers, ..., as he walked across campus, and put them in the next trash bin. Many of you know that he (and I) did not carry through in that vein to our own offices.

My favorite Norman story concerns the fact that he spent long hours in his office. One late afternoon, a beautifully tanned coed knocked gently on his door. He acknowledged her presence and she stated "I can't find my instructor, will you help me with this algebra problem?" He answered "Certainly, as long as you will let me do you a bigger favor!" She asked what that would be? He said, "When we get through with the algebra I want to tell you about the dangers of overexposure to the sun." She did not accept the help.

I will miss Norman. We will miss Norman. He certainly was a unique, intelligent, and unselfish contributor to our community. I will treasure my many memories and the comments that have been made to me about the loss of our friend.

The above photograph was taken during the outing to Stratford, at the TUG annual meeting in Birmingham, England in 1993.

◇ Bart Childs
Texas A&M University
College Station, Texas
`bart@cs.tamu.edu`

Roswitha von den Schulenburg Graham

28 March 1935 – 14 April 1999[†]

Dag Langmyhr

Roswitha worked at the Kungliga Tekniska Högskolan (Royal Institute of Technology) in Stockholm, Sweden. Her job was to organize the production of books, compendiums and other teaching material for the students. She heard about TEX and saw its potential. Even though she never used TEX herself, she became very enthusiastic about it and made a great contribution to introducing it at KTH.

Producing books in the Swedish language, she quickly noted the shortcomings of TEX2. For instance, even though all the letters were there, the Swedish quotation marks were missing. The worst problem, however, was that you could not properly hyphenate words containing either an 'å' or an 'ö'.

In 1988 she was the major force when NTUG (the Nordic TEX Users Group) was founded. Its main purpose was to 'promote the use of TEX and related programs in the Nordic countries (Denmark, Estonia, Finland, Iceland, Norway and Sweden)'. Another important issue was the work to extend TEX to fix the problems of the Nordic users. Members of the Nordic group — and Roswitha in particular — had several meetings with Donald Knuth on this. With the advent of the 8-bit TEX3 and the Cork font encoding, the Nordic languages (and others) got the necessary support.

Roswitha led the Nordic group from its initiation until 1993 and in these years she was also automatically a board member of TUG. After 1993, she remained on the board of NTUG, and her enthusiasm for TEX and the group never diminished.

We who have met her will remember her for this enthusiasm for TEX and its users, but also because she was so immensely hospitable. Quite a few TEX personalities have fond memories of visits to her summer house on a small island in the Stockholm archipelago. In *TUGboat* volume 13(1992) no 4, we can read about Donald Knuth's visit there and the talks he had with Roswitha.

[†] Our thanks to Peter Graham, Roswitha's husband, for providing the photograph.

Roswitha was so very much alive that it is difficult to believe she is no longer among us. We will surely miss her.

◇ Dag Langmyhr
Leader of NTUG
`dag@ifi.uio.no`

You meet the nicest people ...

Mimi Burbank

Everett Larguier, s.j.

In late July of 1998, I received an email message, the first line of which said, "Pardon me for bothering you, but yours is the first contact available ...". The gentleman needed some help with some TEX application on his computer, and kindly provided the necessary information regarding what system, and printer he was using.

What caught my attention was the following:

> *...I am an old man scrambling toward the door of the 90th year of my life, using LATEX and Linux to keep old man Alzheimer from my door.*

I thought then that this man *must* be the oldest TEX user and if not, then certainly one of them!

We have now celebrated the first anniversary of this correspondence,[*] and I must say, "You really meet the nicest people by email." Providing assistance to users can often be a strain on your time, resources and temperament! I can cheerfully say that in this particular case, I've learned nearly as much as any help I've provided. One cannot help but want to know more about someone who is 90 years of age, and using TEX! Sebastian Rahtz became involved in this correspondence and his response to the above statement was, "TEX will keep you young..."

In his own words

> *I am a member of TUG and have been so since about 1989. Reading TUGboat has not been too easy over the years; most of the articles are beyond my comprehension. Over 50 years ago, I got a Ph.D. from Michigan*

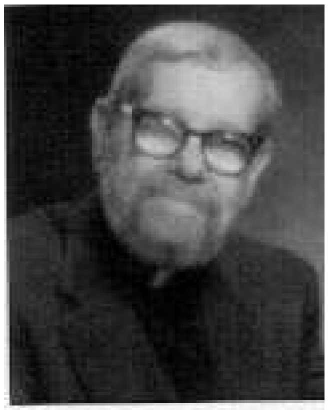
[*] The quotations in this article come from email correspondence as well as from the biographical publication *Peragente Anno Octogesimo Octavo*.

and pursued my professional life as a mathematician at Spring Hill College in Mobile, Alabama, until retirement at age 75. But in being out of the mainstream of mathematical research I have forgotten over these later years much more mathematics than I know now.

In the early '80s, I ventured into using a PC and a VAX as a means of fending off old man Alzheimer. It has been reasonably successful. I picked up a copy of pcTEX along the way, which introduced me to TEX and LATEX. Subsequently, I joined TUG with the thought that it might be helpful. That's where I stand right now. I have been using Linux for a few years now, abandoning DOS, Win95 and the VAX for the most part and becoming a Linux nut.

Needless to say the above only led me to ask more questions, and Fr. Larguier kindly supplied me with a biographical document, *Peragente Anno Octogesimo Octavo*, published by Dragonfly Press, Mobile, Alabama in 1997, typeset using LATEX. The photograph is from an unknown announcement entitled, "Fr. Everett Larguier, SJ, 70 Years a Jesuit". It was very nice to have a "face" to go with the email messages.

Biographical Extracts



Fr. Larguier was born January 26, 1910 in New Orleans, Louisiana. He entered the Jesuits at the age of 19. He attended St. Louis University starting in the fall of 1932, and obtained a Master's Degree in mathematics in 1936. He obtained his doctorate in mathematics from the University of Michigan in 1947, and then served as a faculty member of Spring Hill College in Mobile, Alabama, until his retirement from full-time teaching in 1975. His first publication was in the *Annals of Mathematical Statistics* in 1935, and since that time he has published other mathematical research articles; several books have been published by the Spring Hill College Press.

One of the more humorous moments occurred at the time of his ordination, in June of 1941, at which time it was discovered that his baptism had been recorded to have occurred almost 19 years before his birth!

As we all know, poverty is one of the disciplines of monastic orders, and in reading the biographical material I can only remark that his first job, in 1927, paid $.50 an hour, and he worked about 50 hours a week — surely good preparation for the salary of a Jesuit!

Attendance at mathematical society meetings afforded him the opportunity to meet John von Neumann, one of the most outstanding mathematicians of this century. von Neumann was "impressive by his casual demeanor and lack of pretension. In fact he looked more like a small-town banker than a world-famous mathematician. Perhaps in this respect he was following a family tradition; his father was a banker in Hungary."

Advent of computers

In the late 1970s, Fr. Larguier had an "on-campus" terminal connection to the college computing facilities. Following a move of the Jesuit community to an off-site location, and because of advancing years and increasing arthritic problems, ambulatory access was a problem, and terminal access was provided in his residence. This was later followed by microcomputing facilities — a Zenith computer, Epson printer and modem connection to a VAX computer. He began working with TEX in the l980s.

Fr. Larguier's computing facilities have changed over the years, and he has been gaining experience with Unix and Linux. He says that this was "putting a strain on the brain cells. However as long as some progress is being made in gaining experience with Unix-like stuff, I will know that Alzheimer is not hanging around in the entrance-way to take over my brain." I can only say, "More power to you!"

Today

His interest since retirement has largely been in the area of topology, as well as a continued interest in the history of mathematics. These days, Fr. Larguier is learning a lot about setting up a Linux system, and installing the **TEX Live** CD, writing letters, and working on a book on topology.

◇ Mimi Burbank
408 DSL
Florida State University
Tallahassee Fl 32306-4130 USA
`mimi@scri.fsu.edu`

Views & Commentary

The `french` Package on and off CTAN

Bernard Gaulle

Editor's note: The following letter was distributed in May 1999 to everyone on the French TeX users list. The views expressed are solely those of the author.[1]

– – ∗ – –

The archivists of CTAN (the Comprehensive TeX Archive Network) have been facing an increasing number of requests from editors and user groups wanting to redistribute, and even sell, CTAN archive contents, including software with specific copyright statements. The recurring question then arises: do these specific copyright statements allow such redistribution, and under what conditions?

In response to these redistribution requests [for CTAN materials], different possible options have been considered, rather like a catalogue of extensions or styles and other products, yielding a synthesized attribute, representative of the features of various copyright statements. Various discussions have taken place, more particularly with those making such requests than with authors [of CTAN material]. Amongst the requesting parties, a majority would like to see the archive distributed freely (in the style of the TeX copyright: everything can be taken and and modified, provided the item no longer bears the same name). This is the sense of "free software", which is certainly popular but which addresses a real need, so my criticism is not aimed in this direction. For some, however, this notion has become a sort of religion, and thus warrants some kind of crusade against all those who don't buy into it. What influence such cyber-crusaders have had, difficult to say. But it is true that most of what's available to read has been about exclusion rather than gathering.

Now, CTAN, by virtue of its name and original intent, has always had the aim of assembling *everything* — developed codes and various tools — that exists in the (LA)TeX world. And that has functioned well until a few months ago. However, under pressure from requests to manufacture CDs, the CTAN archivists have decided to split the archive into two: one "`free`" tree, which can be redistributed without any problem, and a "`nonfree`" tree, for which all sorts of restrictions may exist. Justification for the split has been based on the assumption of legal texts, although no-one's been able to give me a single reference. Richard Stallman, founder of the GNU project and the "free software" concept, affirms that such texts exist and suggested that I should consult a lawyer. For my part, I regularly see CDs distributed free of charge with well-known monthly magazines, CDs which feature many "shareware" products with restrictive copyrights. However, none of these magazines has yet been condemned for unauthorized distribution of software. The claim that only "free" products can be redistributed without special authorization is therefore an ironic statement.

The CTAN archivists (who, for the most part, I truly believe only want to satisify their users) began applying their decision at the end of last April, redeploying software according to these two trees, "`free`" and "`nonfree`". And thus, from one day to the next, the `french` package found itself on the "`nonfree`" side. I therefore had to analyse the situation and ask myself if this was acceptable or not, if I have to change something or not.

The French translation of "free" means, of course, 'without cost' or 'freely' ['without constraints' –Ed.]. If `french` then is placed on the "`nonfree`" shelves, it means either that it has to be charged for or it is being held hostage to restrictions of some kind. However, the copyright statement for `french` has existed for years now, and was indeed originally devised in such a way that anyone could use it freely and however they wished. Only modification and commercial distribution were subject to a few restrictions. After discussing this with the CTAN group and seeing that my views were not being understood, I decided that it was too shocking to see `french` placed in the "`nonfree`" tree and therefore I asked that it be removed.

As a result of this action, my long-standing aim to see `french` always available to everyone can no longer be achieved and so I have to ask myself some questions. In the first place, is this CTAN policy of favouring "free" redistribution [of the archive contents] via CD going to last? It's possible but still, I do believe other groups will choose to return to the previous situation and propose an RCTAN (Really Comprehensive TeX Archive Network), in which case everyone would again be happy.[2] If it doesn't happen, then maybe I should consider another form for `french`, more liberal in its rights

[1] The author wishes to thank Barbara Beeton and Christina Thiele for their efforts in finding a translation that makes sense in English without violating the original French. In case of doubt, the original French text is the definitive one: "À propos de french", *La Lettre GUTenberg* 15 (1999), p. 16.

[2] RCTAN has now become a reality: `ftp.loria.fr`.

statement but then also probably more restricted in functionality ... The future and your comments will help shape my choice.

In the meantime, I've chosen freedom, freedom to choose where `french` will be placed, outside the slightly shameful world of "`nonfree`", so that everyone can freely do what they will with it, within the limits of its copyright statement, without *a priori* constraints or commercial connotations. Thus, the `french` distribution will remain available, as always, from the GUTenberg server `http://ftp.gutenberg.eu.org/pub/gut/french`. Anyone may fetch it for free and freely make use of it.

I thank you for having read my text to the end; I have tried to be as balanced as possible because I don't want any polemics. Rather, I hope that all needs can be satisfied in the future, leaving authors free to choose the terms of their copyright statements.

◇ Bernard Gaulle
Vice-President, GUTenberg
`gaulle@gutenberg.eu.org`

— — ∗ — —

Response from the CTAN team

The CTAN team has made the following statement about the content and arrangement of the archive:

> The aim of the CTAN team is to make CTAN consistent, simple, and reliable, both for users and maintainers. We apologize if our policies cause upset to some people.

— — ∗ — —

Editor's commentary

Having been party to some of the discussions that led to the segmentation of CTAN, I understand the intent of the split in a way that is probably somewhat different from that of someone coming upon it *de novo*.

One of the driving requirements for the split was a request to the TEX Live team for permission to distribute the CD beyond the confines of the formal TEX user community, in particular, to include the CD in a commercially published book on LATEX.

Although CTAN contains shareware and tools that originated outside the TEX community, these items are made available by their authors or primary distributors on other net-based archives, and their presence on CTAN is a convenience.

Earlier versions of TEX Live were not much concerned with formal permissions from the authors or primary distributors of the files included on the CD; their presence on CTAN was considered tacit permission, and besides, the intention was to distribute the CD only to the user groups that cooperated in its creation. However, with the request to redistribute TEX Live 4 beyond this limited sphere, permissions suddenly became very important.

For a few items restricted by the originators from wider distribution, special permission was requested, and, in most cases, granted; a special version of TEX Live 4 was generated for the "external" distribution, omitting any items for which restrictions existed and no permission was forthcoming.

In order to make the creation of TEX Live 5 and future editions more straightforward, it was decided to make the provenance of all CTAN holdings obvious without having to check each file. The concept is clear; the naming is perhaps not so clear.

The terms "free" and "nonfree" are short and easily remembered, but "nonfree" seems to imply a monetary transaction. In the CTAN sense, however, it means only that the author has placed some restriction that limits redistribution. This could be a request for a shareware fee, or a statement that a package requires special permission if it is used for other than strictly personal use. In the case of `french`, there is a requirement that any file in the package with an explicit copyright statement not be modified, and the package may not be redistributed as part of any commercial offering regardless of whether or not compensation is asked; these are not unreasonable requests, but they do attach "strings" to the package that mean it cannot be automatically included on a CD such as TEX Live, which may find its way into distribution beyond the user groups.

Perhaps "restricted" and "unrestricted", or (more colloquially) "strings" and "nostrings" might have been better choices of terminology: it's not instantly clear what the terms mean, and if one checks, one will learn exactly what is meant. The kinds of restrictions placed on CTAN offerings are not shameful, and there are good reasons for them in most cases; the CTAN team, as I see it, is merely trying to comply with the wishes of the owners.

I worked for a number of years in international standards working groups. International standards have a reputation for stilted and overly precise language. However, a central requirement for these documents is that they be translatable into many different languages with no change of meaning. This is the misfortune that has now befallen CTAN — an intention to make clear to users that certain items should be checked for possible restrictions has been badly misunderstood.

◇ Barbara Beeton
`bnb@ams.org`

Letters

Letter to the Editor

Jonathan Fine

The good name of TeX

One of the many wonderful things about TeX is that its behaviour is essentially the same, no matter where it runs. TeX is a fixed point, identical on all machines. The same goes for METAFONT and the Computer Modern fonts.

The author of TeX, Donald Knuth, has made it perfectly clear that he does not object to anyone revising TeX(or METAFONT) just as long as the resulting program is called something else. However, he also says "nobody is allowed to call a system TeX or METAFONT unless that system conforms 100the TRIP and TRAP tests".

He also asks us "to help enforce these wishes, by putting severe pressure on any person or group who produces any incompatible system and calls it TeX or METAFONT or Computer Modern — no matter how slight the incompatibility might seem". (Both quotations are from *TUGboat*, 11(4), p489, reprinted in Knuth's *Collected Papers in Digital Typography*).

In a recent article in *TUGboat* (issue 19(4), p366–371), Petr Olsak describes encTeX, a not completely compatible revision to TeX. It seems to me that Olsak has not followed Knuth's wishes in a consistent manner.

Although he calls his new program encTeX, in his article he talks about TeX this and TeX that when he is referring not to Knuth's TeX, but to his own encTeX. For example, he describes the creation of encTeX the program as a "new compilation of the TeX binary" (p367), and throughout the article he talks of iniTeX when in fact he means iniencTeX. On page 369 he writes "the production version of TeX" when in fact he is referring to his encTeX program.

Olsak is tackling a real problem faced by TeX users in his own country, and he deserves credit for this. His solution requires an incompatible revision of TeX the program. If this has to be, it has to be. But more care is required in the documentation.

◇ Jonathan Fine
203 Coldhams Lane
Cambridge, CB1 3HY
United Kingdom
`fine@active-tex.demon.co.uk`

Reply

Petr Olsak

Jonhatan Fine wrote a little response to my article about encTeX published in *TUGboat* 19(4). He is right in all his arguments. My article was written about an extension of TeX, not about TeX itself. This extension was called encTeX. The banner was changed. It was my mistake that in some sentences of my article I talk about TeX but I mean my encTeX extension. Please accept my apology for this. My sentences might add to the confusion about the "name of TeX" for some readers. Jonathan Fine is an example of one such reader. I am sorry.

The primary aim of my article was to show that the correct localisation of TeX in our country is possible only if some extension which is incompatible with the TRIP test is done. The non standard `xord/xchr/printability` settings are explicitly needed. No matter if these settings are implemented via encTeX, via TCP tables in emTeX, via TCX tables in web2cTeX, or constant settings are made in some sections of `tex.web/tex.ch` signed as "system dependent". The resulting program (usable for our localisation) is impossible to call TeX because the TRIP test explicitly specifies that codes higher than 127 are written in two-circumflex notation into `\write` files and logs.

This feature (incompatible with TRIP) is implemented into some widelly used "TeX" distributions: emTeX (with -8 parameter) or web2cTeX (if TCX tables are used or locales are installed and set). This described behavior of web2cTeX implies that it is impossible to call web2cTeX "TeX" if TCX tables are used or locales are installed; yet this distribution is widely known as a TeX distribution. The banner is unchanged. This program is distributed on CDs to all TUG members with the name TeX. This represents more of a problem of the "good name of TeX" than the name-confusion in my article.

In addition, there is the more incompatible extension of web2cTeX from Knuth's original TeX. I mean its sensitivity on first special line in the `.tex` source of the document. If the first line of the document starts with "%&" double, then the web2cTeX switches to behavior undocumented in Knuth's *Computers & Typesetting* (namely volume A and B). I mean, the web2cTeX is not TeX.

◇ Petr Olsak
Faculty of Informatics
Masaryk University
Botanicka 68a, CZ-60200 Brno
Czech Republic
`olsak@math.feld.cvut.cz`

Typography

Typographers' Inn

Peter Flynn
University College Cork

Reversed quotes

The pox of reversed quotes (‛) continues to spread. If it isn't checked, we'll have an entire new generation of typographers who believe that ‛this' is the right way to implement 'quotes'.

I don't know if it's attributable to ignorance, or if there is a version of some lesser breed of software out there which has implemented it as the default. Perhaps someone who is a more frequent user than I of *Q—k X—s* or *F—r* and other such tools could check this and let me know.

The glyph is in itself harmless, and as I said when I started this campaign, I first saw it in a book published in the 1970s, so it has a long and dishonorable history. It would be nice to think that its use just displays ignorance or carelessness, but typographers are usually neither ignorant nor careless: there is some deliberate behavior at work, and it goes unnoticed by most readers. I suspect it comes from a mistaken desire for a spurious symmetry rather than anything else, but what mystifies me is how it came to exist in the first place, and where people are finding it in fonts.

The usual pint in Vancouver or other suitable venue for the person who mails me the grossest example of this abuse.

Tag abuse

Talking of abuse, I shall shortly be relaunching the Society for the Definitive Abolition of Tag Abuse (SDATA). This worthy organisation was set up to campaign for better markup languages in order to relieve authors and editors of the need to abuse existing markup systems, and thus to prevent the more obvious typographic mistakes which arise from ambiguous or meaningless markup.

Tag abuse takes several forms, depending on the language, but the most obvious example perpetrated in LaTeX is the use of `\emph` to achieve italics even when emphasis is not the objective. To some extent this is a problem of our own making: for so long we thundered at the poor users 'Thou Shalt Not Use `{\it }` For Emphasis, Only `{\em }`' that many of them now believe they will be shot at dawn for using the undistinguished `\textit` for italics of any sort instead of `\emph`. The labs are full of them, and they propagate the myth to every new intake of users.

There is admittedly the advantage to `\emph` that it handles its own context-sensitive font control, appearing in italics within a roman body and in roman within italics, but as this is merely a macro in `latex.ltx`,

```
\DeclareRobustCommand\em
  {\@nomath\em
   \ifdim \fontdimen\@ne\font >\z@
          \upshape \else \itshape \fi}
```

I see no reason why it shouldn't be called something like `\romital` and made available for anyone to implement in any circumstance where a context requires a distinguishing font shape.

Despite this kind of misunderstanding, we have been shielded to a large extent from some of the horrors of undistinguished markup: there is far worse outside the TeX world. One system I have seen provided perfectly sensibly for `emph` but covered itself by also providing for `emph1`, `emph2`, `emph3`, and so on, with notes in the specification saying which one was to be used for italics, bold, bold italics, small capitals, etc. This allowed the markup to reflect how the editors wanted the text to appear, but didn't let them specify it meaningfully. They still had to make the decision on which font to use but could not name that reason in the markup.

The whole area of generic markup and meaningful names for things is a two-edged sword for designers. If an author or editor marks some words in italics in a document, does she mean italics *ruat cœlum* (come what may), *or does she mean italics* mutatis mutandis *(according to sense)*? The point about markup abuse is that she shouldn't be marking italics in the document at all in this case, but something like `\foreign` instead, and leaving the font decision to the designer in the stylesheet.

We have become so used to commutative font specification, where the surrounding font parameters are inherited, that users now *expect* to be able to get bold italic small cap sans-serif outline swash characters when requested, and it's no use telling them that the font designer only drew swash characters for a few decorative italic capitals. Worse, many DTP systems actually make a feature of providing any permutation of anything vaguely font-like on command.

On the other hand, there's nothing wrong at all with marking decorative italics or bold for what they are, depite the screams of protest from the purists. What's wrong is calling them something that they are not, like 'emphasis'. I've had users ask me how they can make italics 'more italic' because they want increasing levels of emphasis.

As the world is poised to start the slow move away from hard-coded appearance to a more extensible markup system (well, that's the theory, anyway) it is going to become more important that typographers and compositors are able to untangle the mess left by well-meaning authors or editors unknowingly abusing what they believe to be usable markup. Join now and maybe we can educate them: `http://www.ucc.ie/sdata`.

Word-swallowing

The other day I was explaining to someone who wanted a mathematics textbook typeset that there were only a handful of fonts which included the mathematical symbols and had math-spaced italic characters (Times, Lucida, Computer Modern, Concrete, and another whose name escapes me at the moment). I must have been less than lucid, because he went away with the idea that LaTeX could only set in these five fonts!

No damage done, as I was able to explain that LaTeX could typeset in pretty much anything that was available in PostScript, Metafont, or TrueType: it was only math typesetting that was restricted to the brave few.

The question then arose, could a different body font be used with one of them? Many of you will have seen the effect of setting text in Times and math in CM, and it's not very pretty, but in this case the math turned out to be less complex than usual, as the book is a remedial work for those who succeeded in skipping math earlier in life. With the concentration on arithmetic and simple fractions, and relatively few symbols beyond $+$, $-$, $\times$, and $\div$, it's perfectly possible to get away with pretty much any suitable book font, such as Palatino.

However, the pretty little PostScript font installer I mentioned last time has taken a major nosedive. An MS-Windows crash led me to give up my last Microsoft machine and return to Unix, so while I still have the source code for the font installer, I don't willingly have the platform, and a lot of people have mailed me to ask when it will be available.

The program was written in response to the large number of complaints and requests I get about difficulties in installing Type 1 fonts for LaTeX systems. Like many long-term users, I spend a small but significant amount of time explaining to others that LaTeX is not restricted to CM — math mode or not — and Type 1 is still the easiest of the other formats to handle[1] and the typographic facilities provided by pstricks are too useful to pass up. However, too many install-time options, especially for font encodings, made me realise that what most users want is a simple, prescriptive installer which you point at a directory or CD-ROM of fonts and tell it to install selected fonts come hell or high water and not to go asking questions.[2] In its last incarnation it not only did the `.afm` to `.tfm` conversion and file-copying, but added the relevant line to `psfonts.map`, and created the `.fd` and `.sty` files in `..\latex\local`, and ran texhash (or equivalent) to update things (it did assume that the user's installation was TDS-compliant, however).

It was written as a pilot in *Visual DisplayScript* for Windows, which was the only tool I could find at the time with anything like the functionality needed for writing simple windowing utilities, and I mentioned that I was seeking a similar environment for Unix. Several readers pointed me at *Tcl/Tk*, which I was vaguely aware of from earlier attempts but had never managed to get working. The recent versions are hugely improved, and as it is multi-platform, the rewrite of the font installer will be available for Macs, MS-Windows, and X. The bad news is that because of this shift, it won't be in a usable form for Vancouver, so a large helping of humble pie is my dessert. Sorry.

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

[1] I have had no success in getting TrueType fonts to work in LaTeX: if someone can point me at a reliable, authoritative, prescriptive, and bug-free document describing the procedure, I'd be very grateful.

[2] The default encoding I use is Y&Y's LY1, for the simple reason that it's the only one I've found which puts all the characters I want in places where LaTeX and dvips can find them: if someone can point me at a reliable, authoritative, prescriptive, and bug-free document describing why another encoding is superior, I'd be very grateful.

Fonts

A short introduction to font characteristics[*]

Maarten Gelderman

Abstract

Almost anyone who develops an interest in fonts is bound to be overwhelmed by the bewildering variety of letterforms available. The number of fonts available from commercial suppliers like Adobe, URW, LinoType and others runs into the thousands. A recent catalog issued by FontShop (Truong et al., 1998) alone lists over 25 000 different varieties.[1] And somehow, although the differences of the individual letters are hardly noticable, each font has its own character, its own personality. Even the atmosphere elucidated by a text set from Adobe Garamond is noticably different from the atmosphere of the same text set from Stempel Garamond. Although decisions about the usage of fonts will always remain in the realm of esthetics, some knowledge about font characteristics may nevertheless help to create some order and to find out why certain design decisions just do not work. The main aim of this paper is to provide such background by describing the main aspects that might be used to describe a font.

The outline of the remainder of this paper is as follows. First I will discuss some basic font characteristics. Next some elementary, numerical dimensions along which properties of a typeface design can be assessed will be discussed. The next section elaborates on those measures and some additional aspects of 'contrast' will be discussed. The final two sections briefly present a font classification along the dimensions discussed in the previous section and some implications.

Some elementary differences

Proportional and monospaced. A first difference that can be recognized between typeface designs is the spacing of fonts. Monospaced or typewriter fonts in which each character occupies the same amount of space can be distinguished from proportionally spaced fonts.

`Computer Modern typewriter (monospaced): Winmvw`

Computer Modern Concrete (proportionally spaced): Winmvw

Hardly anyone will dispute the statement that proportionally spaced fonts are more beautiful and legible than monospaced designs. In a monospaced design the letter i takes as much space as a letter m or W. Consequently, some characters look simply too compressed, whereas around others too much white space is found. Monospaced fonts are simply not suited for body text. Only in situations where it is important that all characters are of equal width, e.g., in listings of computer programs, where it may be important that each individual character can be discerned and where the layout of the program may depend on using monospaced fonts, can the usage of a monospaced font be defended. In most other situations, they should simply be avoided.

Romans, italics, and slant. A second typeface characteristic that will hardly be new for any TEX-user is the difference between italic, oblique (slanted) and roman fonts. The difference between italic fonts and the roman fonts lies in their history. Italic fonts are the descendants of handwritten letter shapes, whereas the roman fonts were originally chiselled in stone. Consequently, the romans look more rigid; the italics, to the contrary, show more elegance and are more 'curvy'. Furthermore, the shapes of some individual characters differ; this difference is most apparent when we look at *a*, *g* and a, g (here in the italic and roman variant respectively). The origins of the italics being in handwriting, they are usually slanted, whereas the romans are typically typeset upright. This, however, is not strictly necessary. Italics can theoretically be typeset upright and romans may be slanted:

An upright italic and a *slanted or oblique italic*

An upright roman and a *slanted or oblique roman*

Generally designers agree that text set in roman is more legible than text set in italic, although the readability of italics accompanying different fonts may differ considerably, which is important if large pieces of text are typeset in italics. Compare for instance:

[*] Apart from some minor modifications, this article is identical to an earlier publication in MAPS, the communications of the Dutch TEX User Group, Nummer 22, Voorjaar 1999, pp. 81–93.

[1] This enormous variety is partially made possible by the introduction of electronic typefaces, which allows for worldwide distribution without exceptional cost. In 1950, that is before the advent of electronic typesetting, Groenendaal could still attempt to list *all* typefaces readily available to an ordinary typesetter.

A block of text set from Utopia Italics. Generally designers agree that text set in roman is more legible than text set in italic, although the readability of italics accompanying different fonts may differ considerably, which is important if large pieces of text are typeset in italics.

A block of text set from Computer Modern italics. Generally designers agree that text set in roman is more legible than text set in italic, although the readability of italics accompanying different fonts may differ considerably, which is important if large pieces of text are typeset in italics.

If multiple slanted fonts are used in one piece of running text, it is important to ensure that the angle of slant is comparable, otherwise a page will look rather uneven.

Serif and sans serif. An issue that raised much discussion in the first half of this century (see e.g., Tschichold, 1991) but on which a *communis opinio* now seems to have been reached is the usage of serifed or sans serif fonts:

Computer Modern (with serifs)

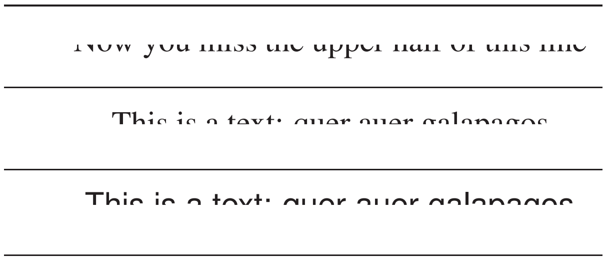
Computer Modern sans (sans serif)

Whereas at the beginning of this century a large group of designers were of the opinion that sans serif designs were to be preferred as they were more modern, emphasizing the pure shape of the individual characters and omitting superfluous elements, it is now generally recognized that the serifs have an important function for the following, not always independent, aspects of legibility:

- Serifs make individual characters more distinct. In their sans serif variant many characters look remarkably, if not exactly, like mirror images of each other. During the reading process they are easily confused, especially by persons suffering from dyslexia. The advantage of serifed typefaces over their non-serif counterparts, in this respect, is easily seen from the following example:

  b d   b d
  p q   p q

- Serifs emphasize the beginning and ending of individual characters, compare e.g., rn with rn.
- Serifs emphasize the shape of words. It is generally recognized that experienced readers do not read individual characters, but read words and mainly use the upper half of a line of text for this purpose. The general claim is that the serifs facilitate this process. Just check it for yourself by looking at the next set of lines:



Furthermore, serifs have an important function in shaping the personality of a type design. Different serifs — a set of possible serifs is presented in Figure 1 — give a typeface design a clearly distinct personality.

The first serif actually is no serif at all. The second one, the slab serif, is orthogonal to the stem to which it is attached and has about the same width as this stem. Slab serifs are generally, but not necessarily (Lucida Typewriter is a well-known example), used for monospaced fonts like Courier and Computer Modern Typewriter. Some proportionally-spaced fonts, like the Computer Modern Concrete we encountered earlier in this paper, also have slab serifs. Those fonts are generally called Egyptiennes and are normally used for two purposes: display text in advertising, and typesetting labels on maps. A well known example is 'Atlas', by the Amsterdam Typefoundry (see Figure 2). An important reason for using slab serifs in this latter type of copy may well be that the serifs clearly belong to the letters and consequently are not likely to be confused with other elements on the map.[2]

The next type of serif, the wedge serif, has been popular in advertising and for book covers during the fifties and sixties of this century, but is hardly used nowadays. The main, and probably only, advantage of this design is that is is easily drawn by hand and still looks somewhat unusual.

The hairline or modern serif is typical of 'modern' typefaces like Didot or Bodoni (see Figure 3). Such serifs became popular in the second half of the eighteenth century. Great craftmanship was required to make the matrices needed to cast letters with those extremely thin serifs. Furthermore, great care had to be taken during printing, as the hairline serifs were very fragile and could easily break.

[2] A second reason for the preference for Egyptiennes and sans serif fonts in applications like map printing is that the contrast of those fonts typically is near unity; see the discussion on contrast later in this paper.

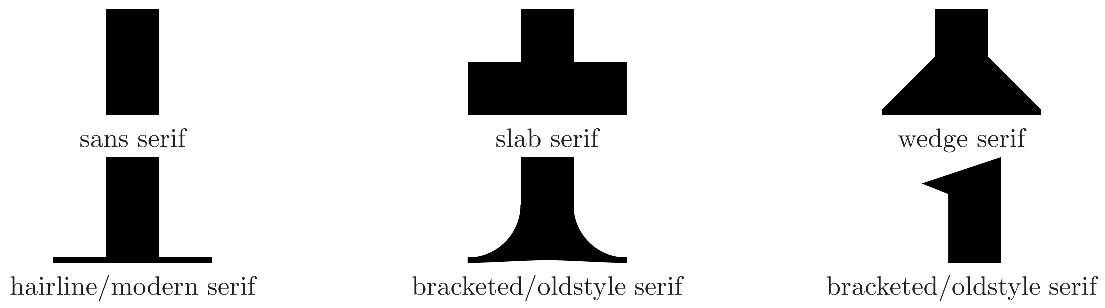


Figure 1: Different types of serifs.

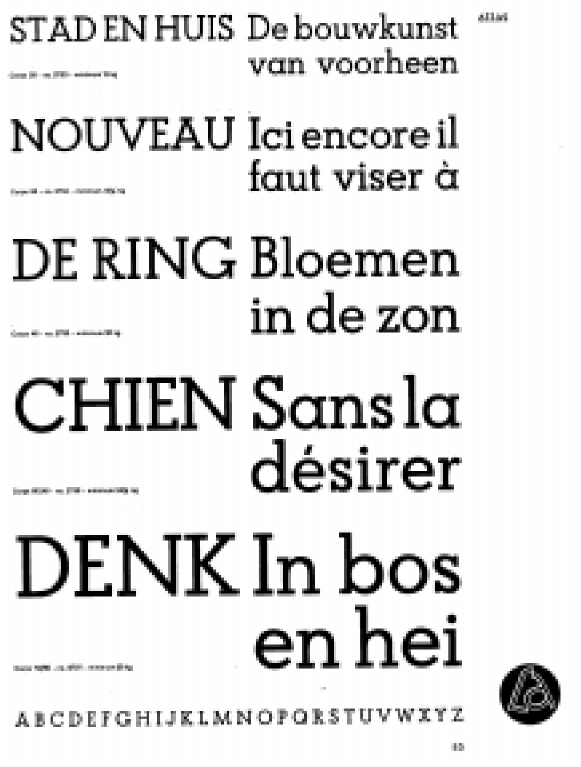

STAD EN HUIS De bouwkunst van voorheen
NOUVEAU Ici encore il faut viser à
DE RING Bloemen in de zon
CHIEN Sans la désirer
DENK In bos en hei
ABCDEFGHIJKLMNOPQRSTUVWXYZ

Figure 2: Font specimen of 'Atlas' (source: N.V. Lettergieterj Amsterdam [Undated]).

Nowadays, one sometimes wonders whether those designs are the equivalent of Paganini's capriccios for violin, if their main purpose is not to show craftsmanship rather than beauty? Nevertheless, one has to admit that a book in Bodoni, carefully typeset on the right kind of paper, still looks stunning (apart from blackletter, Bodoni is one of the very few typefaces that looks good in combination with high contrast illustrations like woodcuts Groenendaal, 1950).

The serif we encounter most often is the bracketed or oldstyle serif (both the lower and upper serif are shown in Figure 1). This is the traditional serif, found in fonts like Garamond, Bembo and Times.[3]

The dimensions of a typeface design

Size and design size. The best known, and probably least useful, dimension of a font is its 'size'. Everyone has encountered remarks like 'this text is set from a 10-point Bembo' and 'papers should be submitted in 12-point Times Roman'. Traditionally the size of a font is the height of the piece of lead from which the text is set. Nowadays the size of a font can generally be considered an almost useless figure. In most fonts it is equal to the height of the parentheses ('()'), but even that is not always the case. In wordprocessors, the point size will generally be equal to the distance between lines of text if you set linespacing to one. For practical purposes this knowledge is limited; the only thing about font size that is important is that most fonts have a design size. This is the size at which the font will look best. Although bu using modern typesetting software like TEX, or any Windows or Macintosh program, it is possible to scale a font to any desired size, you will generally get better results if you stick to a size in the neighbourhood of the design size. For some popular fonts, like Times Roman or our good old Computer Modern, different design sizes even are available. This allows the careful designer to use all fonts at their optimal sizes. When using Computer Modern, the standard LATEX document classes even take care of this automatically: the footnotes, for instance, are set from a font with another design size than the font used for the main text. This ensures an equal level of 'grayness' across the page and increases legibility (characters of fonts with a smaller design size are generally somewhat wider and heavier); look for instance at the difference between the next two examples:

[3] Times is somewhat peculiar in this respect: the bold characters use modern serifs, the ordinary roman, oldstyle serifs.

Computer Modern with
5-point design size

Computer Modern with 17-point design size

The x-height. For practical purposes, a more important characteristic is the x-height of a font, which is exactly what the name implies: the height of an x (or any other letter without ascenders or descenders) in the given font.[4] The x-height of a font essentially determines the size of the font as it will be perceived by the reader.Fonts with an identical nominal size may have x-heights that differ surprisingly. The next two examples show Utopia and Garamond at the same size. The x-heights, and consequently the perceived size of the font, however, differ considerably:

Hamburgefont Hamburgefont

When combining fonts in running text, for instance when using typewriter or sans serif fonts in combination with an ordinary serifed roman, it is important to ensure that the x-heights of all fonts used are identical. A traditional problematic combination consists of the standard PostScript fonts Times, Helvetica and Courier. Those fonts have quite different x-heights, which distorts the evenness of a page if no measures are taken:[5]

Times Helvetica `Courier`

Fortunately, the New Font Selection Scheme (a short introduction to the NFSS can be found in Kroonenberg, 1999) makes solving this problem rather easy: the default is to load each font at the same size; however, it is also possible to specify a scale factor in addition, which may be used to compensate for different x-heights.

Ascenders, descenders and capitals. In addition to the x-height and font size, three other height-related dimensions of a font are available: the height of the capitals (e.g., K, H, and S), the height of the ascenders (e.g., k, l, and h), and the length of the descenders (e.g., j, g, and y). In many fonts the capital-height is equal to the height of the ascenders; sometimes, however, the ascenders are slightly longer than the capitals. The main advantage of making the capitals slightly shorter than the ascenders is that this gives a more even level of grayness across the page; otherwise — especially when the ascenders are large relative to x-height — the capitals would stand out too much.[6] An example of a font that uses slightly smaller capitals than ascenders is Garamond:

HhKkLlAk

The combination of x-height and ascender and descender heights roughly determines how economical a typeface is,[7] in other words: how much text can be put on a page without sacrificing legibility. Fonts with relatively large x-heights compared to their size can be used at small sizes. Consequently, they are rather economical: more lines of text can be put on a single page and more text will fit on a single line. However, the gain is not as large as one might hope for: fonts with relatively large x-height generally require some additional interline spacing.

Width and stem width. Apart from the measures of font height, discussed in the previous paragraphs, we also need some measure of font width. TEX provides the user with an amount called em-space, the width of a single m, which for design considerations has relatively little importance. Somewhat more important is the average width of a font, generally measured (Rubenstein, 1988) by the total width of all lowercase characters. This width is also of importance when combining fonts. Although less perceptible than the x-height, fonts with different widths (given an identical height) tend to combine badly (this problem is mainly related to the 'rhythm' of the font, to be discussed later in this paper).[8] Of course width also is related to the amount of text that can be put on a page; the larger the width the smaller the number of characters that fit on a single line. Not surprisingly, fonts with an x-height that is relatively large tend to have a large width as well, thus reducing the economy gained by using such a font.

A final directly measurable characteristic of a font is stem width: the width of the stems of letters like l. Of course this also influences the results when combining different fonts in a piece of text. The next example shows two monospaced fonts, along

[4] The x-height of a font is readily available in TEX. If you want to specify a length in terms of the x-height of the current font, just use the measure ex, instead of a more traditional measure like cm or pt.

[5] The example also shows that color and rhythm of the three typefaces differ.

[6] Barbara Beeton drew my attention to the fact that this is especially important when typesetting text in German, where every noun is capitalized.

[7] Morison (1997) even claims that the general principle behind the evolution of font design is economy, and indeed more recently developed typefaces tend to be more economical than traditional ones.

[8] Unfortunately TEX is only able to scale the height and width of a font simultaneously, so this problem is not easily solved. Future generations of TEX may well solve this problem.

with a Times. With regard to stem width (and consequently blackness) Computer Modern typewriter combines far better with Times than the traditional Courier (but of course, the x-height still needs some adjustment).

`Courier` Times `Computer Modern Typewriter`

Some more complex dimensions

Color. Although it is impossible to characterize a font completely by a set of numbers, we may refine the measurements presented till now to get some additional insight into the properties of a design. Most TEX-users, for instance, will have heard the remark that Computer Modern is 'too light'. This somewhat subjective criticism can be made more objective by calculating a measure of 'color'. This measure is defined as the ratio of the width of the set of all 26 lowercase letters, divided by the stem width (Rubenstein, 1988). In other words, color is a measure of the amount of paper left white: the higher the color-value of a font is, the lighter it looks. Color values for a number of popular fonts are provided in Table 1. It is evident that Times, which is the font of reference for most people, is much darker than the Computer Modern fonts. What also is noteworthy is that the 12-point Computer Modern is somewhat lighter that the 10-point variant. Finally, one may notice that, notwithstanding the common criticism that Computer Modern is 'too light', it is not the lightest font in the small set presented here: Garamond is even lighter. Apparently, color is not all there is to say. When we look at the other measures provided in this table, it seems as if Garamond is able to compensate for an apparent lack of color by a high contrast value.

	color	*contrast*	*weight*
cmr12	197.111	1.703	0.146
cmr10	192.258	1.650	0.153
Times	156	2	0.17
Garamond	208	3	0.15
Helvetica	163	1	0.16
Bembo	184	2	0.16
Van Dijck	191	2.75	0.15

Table 1: Color, weight and contrast of some popular fonts (the statistics for Times, Garamond, Helvetica, Bembo and Van Dijck are based on measurements presented in Rubenstein (1988); the statistics for both Computer Modern variants were kindly provided by Taco Hoekwater).

Contrast. Contrast is defined as the ratio between the width of vertical and horizontal stems (Rubenstein, 1988). Contrast is, roughly speaking, what makes a font lively, brilliant if you wish. If contrast gets extremely high, a font is hardly legible at all and only suited for use as a display typeface in, for instance, advertising. Similarly, fonts with extremely low contrast are hardly legible. Endless discussions about optimal contrast values are, of course, possible, but there seems to be some general agreement that for, serifed typefaces, contrast should be somewhere between 2 and 3.5. It is evident from the data presented in Table 1 that Computer Modern scores rather low on the contrast (of if you wish, high in the 'dullness') dimension. The design simply lacks contrast to an extent that may impel legibility. The cautious reader may also have noticed the extremely low contrast value of Helvetica. Such contrast values are rather typical for sans serif typefaces, which tend to stress evenness, often at the cost of legibility.

There is another aspect of contrast that deserves attention: contrast also is an indication of the 'fragility' of a font. At low resolutions (or looked at from large distances) designs with high contrast may be seriously distorted. This is one of the main reasons why sans serifed typefaces (and typewriter and slab serif fonts, which also tend to have contrast values near one) are the fonts of choice for transparencies, traffic signs and computer displays.

Theoretically, contrast values between zero and one are also possible. Such extreme designs, however, are only suited for advertising and other more-or-less artistic utterances.

Weight. A final, common dimension of a font is its weight. Color measures the darkness of a font as it appears to the reader who looks at a page of text. Weight is used to assess the darkness of the individual letters and it calculated by dividing the vertical stem width by the x-height of the font. According to Rubenstein (1988) if weight lies outside the range 0.15–0.2, legibility suffers. Apart from the 12-point Computer Modern all fonts presented in Table 1 are within this range. Times is the most 'weighty' design in the set of fonts presented here, but the differences are less noteworthy than on the previous dimensions.

Additional aspects of contrast

Contrast is one of the more important aspects of a type design. However, the measure of contrast presented above does not cover this aspect completely. A first additional aspect of contrast is the axis of

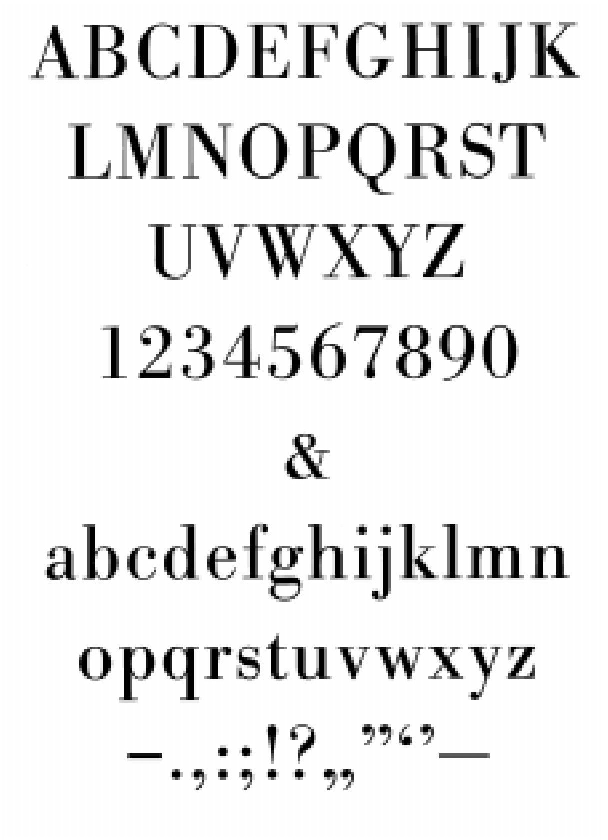


Figure 3: Font specimen of 'Bodoni' (source: Klein et al., 1991).

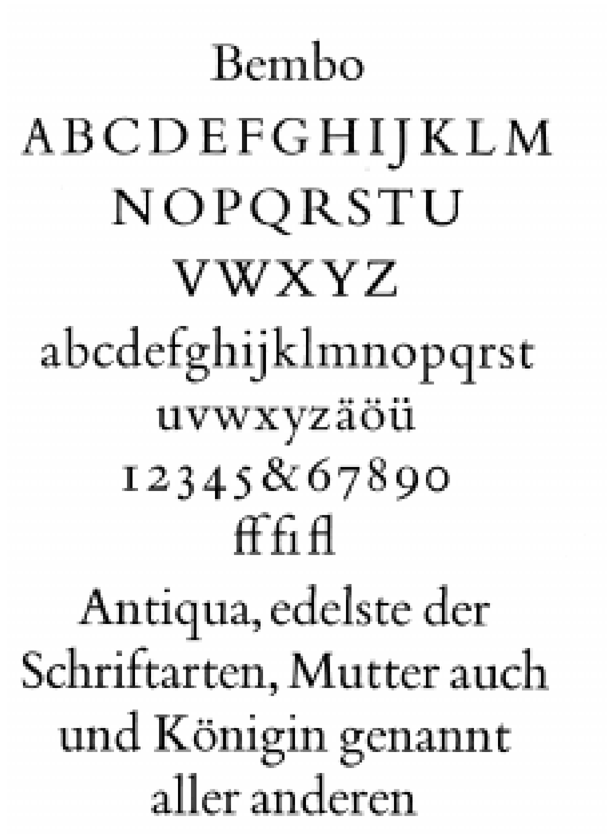


Figure 4: Font specimen of 'Bembo' (source: Tschichold, 1992).

contrast, or the angle at which the broader parts of the characters appear. If we compare, for instance, the design of Bodoni (see Figure 3) with Bembo (see Figure 4), it is not only clear that contrast of Bodoni is higher than that of Bembo, but also that the axis of contrast differs. This is most easily seen, by comparing the 'o' or the 'e' of both fonts. In Bodoni, contrast is orthogonal to the baseline, whereas in Bembo, it is slanted to the left.[9] The axis of contrast has little influence on legibility of a typeface, although the axis of contrast is related to contrast and hence influences legibility indirectly.[10]

The second additional aspect of contrast, frequency, is a far more important determinant of legibility. Figure 5 show the sensitivity of the human eye as a function of frequency. Sensitivity is, roughly, defined as the ease with which for instance *individual* lines, drawn on a sheet of paper can be distinguished. If the lines are very far apart, that is frequency is low, the human eye is simply not able to focus on both lines simultaneously and sensitivity is low. If the lines are very close to each other, frequency is high, the human eye does not distinguish individual lines any more. Although a page may contain black and white lines, it is perceived as being gray.[11] The ability of the human eye to perceive individual lines, rather than no lines at all, or some level of gray, is at a maximum somewhere between 6 and 11 cycles per degree. Of course, in order for a typeface design to be legible, it is highly desirable that the individual strokes of the characters are easily discernible. Unfortunately let-

[9] If one mentally imagines the 'o' begin drawn on paper with a broad brush or pencil, the brush would be held horizontally when drawing the Bodoni 'o', and at a 30° angle when drawing the Bembo 'o'.

[10] To maximize contrast, the horizontal parts have to be as thin as possible and this can only be accomplished using a 'horizontal brush'.

[11] Frequency is not defined in terms of lines per inch but in terms of lines per degree of visual angle. If the sheet of paper is closer to our eyes, the number of lines per degree of visual angle diminishes, although the number of lines per inch remains the same. In this way the individual lines that look like uniform gray at reading distance, become distinguishable at closer examination. At a reading distance of about 40 centimeters, frequency in lines per inch is about two times as high as frequency in lines per degree of visual angle.

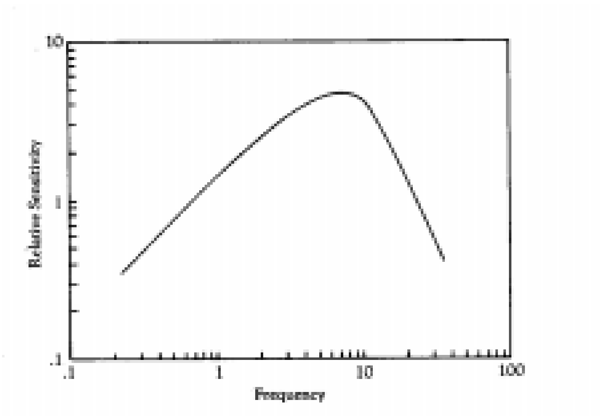


Figure 5: Sensitivity of the human eye as a function of frequency (in cycles per degree of visual angle) (source: Rubenstein, 1988).

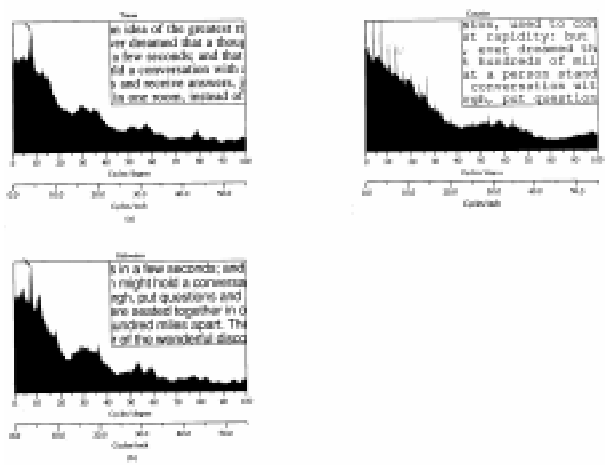


Figure 6: Results (power spectra) of Fourier analysis on text samples in three popular typefaces (source: Rubenstein, 1988).

ters do not consist of simple lines but are slightly more complex: a single number will not suffice to describe the frequency of a font. A number of frequencies will be present on a single page. Fortunately, using Fourier analysis it is possible to find those frequencies and make a plot of them, as is done in Figure 6 for three popular typeface designs: Times, Helvetica and Courier. Now we can look for a dominant frequency which hopefully lies some where between 6 and 11 cycles per degree. The results confirm our expectations: both Helvetica and Times show a clearly distinguishable peak in their frequency distribution at about the point of maximum discernability to the human eye. Helvetica, however, shows a second peak, which will make the design less readible. Courier, finally shows at least four peaks in its frequency distribution.

From characteristics to classification

The characteristics mentioned in the previous section provide the clues that can be used to build a classification of typefaces. The traditional classification scheme distinguishes four categories of serifed typefaces: Venetian, oldstyle, transitional and modern. Venetian typefaces have been in use since about 1470. They are hardly distinguishable from oldstyle typefaces, which have been in use since about 1500. Both categories of fonts share a slanted axis of contrast and the usage of, not surprisingly, oldstyle serifs. Capitals, typically, are somewhat smaller than the ascenders, they end where the serifs of ascenders start. One reason for this is that the ascenders and descenders of those fonts are relatively long and their x-height is relatively small. Furthermore, those fonts are typically relatively light, and contrast is not extreme. To distinguish a Venetian font from an oldstyle font, two features are of importance: first, oldstyle fonts usually have a horizontal crossbar of the lowercase e, whereas this crossbar in a Venetian is at an angle of about 20° with the baseline (like in the ‘Heineken’ logo). Furthermore, the oldstyle capital M has the usual serifs, whereas the Venetian M has double serifs. Prime examples of oldstyle fonts are Garamond, Baskerville and Caslon. Popular Venetians are Cloister, Centaur and many of the designs by Goudy.

The first transitional font was the ‘Romain du Roi Louis XVI’ designed for French governmental publications in about 1702, but only came into general usage at about 1755. Although the serifs of those fonts are already horizontal, the contrast axis is not yet orthogonal to the baseline, but more upright than in the Venetian or oldstyle typefaces. It is generally claimed (Morison, 1997) that the ascenders are as high as the capitals in those transitional fonts, however, upon my examination of some font specimens I learned that this rule is not universally valid. Similarly, although the transitional fonts are supposed to have lining numbers instead of oldstyle numbers,[12] this also is not always the case.

[12] Lining numbers all have the same height and do not have ascenders and descenders. Oldstyle numbers, on the contrary, differ in size and some numbers (e.g., 9) have descenders, whereas others (e.g., 6) have ascenders. Another important dimension along which numbers may vary is whether they are fixed-width or not. This latter aspect is of course important for their applicability in tabular material. Thanks to Barabara Beeton for making this additional comment.

The transitionals are generally blacker than oldstyle fonts; they look stronger, but less elegant.

Finally the moderns, of which Bodoni and Didot are the prime examples, can be found from 1790 on. The development of those typefaces continues the development started with the transitional fonts. The x-height slightly increases and the capitals are as high as (and sometimes even slightly higher than) the ascenders. The axis of contrast now is completely vertical and the serifs are horizontal. Contrast often is extreme, a page set from Bodoni looks brilliant. Although the page may look particularly well from a distance, legibility may suffer from this extreme contrast. Other moderns, like Egmont and Walbaum, are less extreme in this respect and consequently more legible. Table numbers are the rule, but exceptions may still occur.

Some implications

Typefaces, of course, neither were nor are designed with the classification or the numerous characteristics mentioned above in mind. The classification is not perfect, in particular, recently-developed fonts are difficult to classify. As a taxonomy, the classification scheme is useless, it merely functions as a starting point in determining the characteristics of a typeface, and the way it may be used. Typography remains an art, not a science, and each rule has its exception, but some rules of thumb may nevertheless help.

In the previous sections numerous aspects of font selection have already been mentioned. Monospaced fonts are generally not the best choice. Only for typesetting computer programs and similar applications, may they be the preferred kind of typeface. For applications like traffic signs, transparencies, computer applications and other messages that have to be read at low resolution or from a large distance, typefaces with low contrast, particularly sans serif and slab serif typefaces are generally preferred.

For typesetting large amounts of text, e.g., in a journal or a book, serifed typefaces are generally the best choice. If the result has to be striking, modern typefaces are preferred. They may draw attention to a magazine the consumer otherwise wouldn't buy or to a feature article that otherwise might be skipped by most readers. Modern typefaces may also be the font of choice because they blend well with illustrations or emphasize the 'designer-like' atmosphere of a book. Art books are a typical example.[13]

If it may be assumed beforehand that a text will be read, for instance in the case of a novel, oldstyle and transitional designs are preferred. Legibility of those designs is better than that of any other font category. Economy may be one of the criteria for font selection: with transitionals, generally more text can be put on a given amount of paper than with the oldstyle fonts. Oldstyle fonts, on the other hand may be slightly more legible and, more importantly: they look more elegant. Selection of a particular typeface may also be guided by other considerations: Caslon is a fairly appropriate choice for a text by Spinoza; for a French novel from the early 19th century a Didot may be the right choice, just because of the contemporary atmosphere elucidated by such a design.

After a certain typeface has been selected, some general guidelines may be drawn knowing its place in the classification scheme. Again, those guidelines are not laws, but mainly "rules of thumb". With Venetians and oldstyles the œ and æ ligatures may be used, and usage of the fi, fl, and ffi ligatures is almost required. When using a modern or transitional, the f-based ligatures can be omitted, and usage of the other ligatures generally looks kind of overdone.

Font selection for the body text also has some implications for other design decisions. One of the charms of oldstyle fonts is that they look so quiet. To maintain this feature, chapter and section headers may be typeset from an ordinary roman or from small capitals rather than the more commonly encountered boldface variant. In some cases, depending on how similar to the roman font this variant is, an italic may also work. Combined with modern faces, however, a design in which only ordinary roman and small capitals are used looks just too withdrawn. The timidity of such a design just does not mix with the aggressiveness of a modern font.

A final remark may be made about the combination of different typefaces in a single design. Generally speaking it is required that both typefaces are clearly distinct. Furthermore it most often works best when the typeface used for headers and other sparingly used features is blacker than the font used for body text. Thus a Helvetica for section headings with a body text of Times may work well. Bembo for headings with Garamond for the body text (or vice versa) will just be plain ugly. Bodoni for the headings with a body of Garamond may work (if

[13] The majority of the applications in which modern typefaces can be used share another characteristic: they are typically printed on glossy paper which not only combines well with the atmosphere of, e.g., a Bodoni, but also is a prerequisite for adequate printing of the extremely thin hairlines of this typeface.

used with care); Garamond for the headings with Bodoni for the body will probably be ugly, etc. One may feel tempted to deduce the general rule that when combining two typefaces, the least legible one is most suited for headings.

Of course, the rules mentioned above have their exceptions. The only way to find out what works is to experiment. The guidelines given may just help to reduce the number of options to be investigated and to explain afterwards what did and didn't work. And this feature, combined with an urge to communicate the joy that playing around with fonts gives me, was the main aim I had with this article. To anyone who wishes to pursue the topics touched upon in this paper in more depth, I can recommend reading Tschichold's treasury of art and lettering. For those interested in technical details, Rubenstein's monograph is a valuable source book.

References

Groenendaal, M. H. *Drukletters: hun ontstaan en hun gebruik*. De technische uitgeverij H. Stam, 1950.

Klein, Manfred, Y. Schwemer-Scheddin, and E. Spiekermann. *Type & Typographers*. SDU Uitgeverij, 's-Gravenhage, 1991.

Kroonenberg, Siep. "NFSS: using font families in LaTeX 2_ε". *MAPS* **22**, 52–54, 1999.

Morison, Stanley. *Letter forms: typographic and scriptorial*. Hartley & Marks, 1997. Originally published: New York, Typophiles 1968. (Typophile chap bok 45).

Rubenstein, Richard. *Digital typography: an introduction to type and composition for computer system design*. Addison-Wesley, 1988.

Truong, Main-Linh Thi, J. Siebert, and E. Spiekermann, editors. *Digital Typeface Compendium: Font Book*. FontShop International, Berlin, 1998.

Tschichold, Jan. *Schriften 1925–1974*. Brinkmann & Bose, Berlin, 1991. Herausgegeven von Günter Bose und Erich Brinkmann.

Tschichold, Jan. *Treasury of alphabets and lettering*. Lund Humphries, London, 1992. Introduction by Ben Rosen.

◇ Maarten Gelderman
De Boelelaan 1105-kr 3A36
1081 HV Amsterdam
The Netherlands
geldermanm@acm.org
http://www.econ.vu.nl/kw/members/maarten.htm

Fonts

METAFONT: Practical and Impractical Applications

Bogusław Jackowski

The First Steps

This article is intended to be an introduction to problems related to preparing fonts for the TEX system using METAFONT. Many details will be omitted, hence the reader may find quite a large number of intentional or inevitable inexactitudes. I believe, however, that the crucial points can be illustrated by a good number of simple examples.

One should not expect to become a METAFONT expert after reading this article. I will be satisfied if the presented material turns out to be comprehensive enough to teach what METAFONT is, how to use it in simple cases, and in which cases it is most promising to use it.

I assume that the reader knows the TEX system a bit, e.g., what is the name of its author (hint: the same as the name of the author of METAFONT), what DVI files are, how to process documents, what drivers are, etc. In short, I assume that the question "what is TEX" does not require an answer. Instead, I will try to answer the question:

What is all that METAFONT?

First, however, one should ask "What is a font?" For TEX, a font is a collection of data stored in a metric file (TFM). TEX examines it in order to find character codes, the dimensions of characters (height, width and depth), kerns to be inserted automatically between some characters (implicit kerns), etc.

Note that TEX does not care about the shape of characters — only drivers are interested in that. Commonly, the shapes of the characters are stored as bitmaps in PK files or — rarely — in GF files. Bitmaps are not obligatory. If PostScript is involved, outline (Type 1) fonts can be used. Nevertheless, Tomas Rokicki, the author of one of the most popular PostScript driver, `dvips`, and the author of PK coding says that bitmaps are usually more efficient. On the other hand, storing a lot of bitmaps of various sizes for various output devices leads quite soon to storage problems.

A cure is to employ METAFONT for generating the required fonts on the fly. The process of generating the set of bitmaps for a single font of the Computer Modern family on a PC computer with a 486 processor takes a few dozens of seconds. Since the Computer Modern family consists of about ninety fonts, the time needed for the generation of the complete set of fonts is several minutes, which is negligible in comparison with the time needed to prepare a document using TEX.

METAFONT is not merely a program for generating bitmaps. In fact, it is a programming language, resembling AWK, BASIC, C or Pascal. The main difference is that METAFONT is equipped with special tools (data structures and operations) facilitating the description of graphic objects and assembling them into a TEX font. I will focus on these two aspects: first, the graphic capabilities of METAFONT, and second, employing METAFONT for generating fonts. This, hopefully, should answer the question posed in the title of this section.

Whenever convenient, the practical aspects of using METAFONT will be briefly considered. Briefly — because it does not make much sense to theorize about practice; moreover, METAFONT is very simple to use and a few minutes with a moderately experienced METAFONT user is usually enough to master running the program.

How to run METAFONT ?

The description of a graphic object in the METAFONT lingo consists of a series of statements (instructions) to be interpreted and executed consecutively. METAFONT performs calculations and generates the bitmaps of the processed graphic objects (characters) in the form of a GF file (generic file) and a TFM file (TEX font metric file). Additionally, a LOG file is created which contains the information about the run and messages issued by METAFONT during the run.

The programmers' tradition is that we should start with a dull and trivial example. It is a bad custom not to respect tradition, so let's assume that we have prepared the following program (a percent, as in TEX, begins a comment; a semicolon, as in Pascal, ends a statement):

```
message "This is a trivial program.";
end
```

Invoking METAFONT[1] in the following way (please note the name "plain", known from TEX):

```
mf386 \&plain foo.mf
```

will result in producing neither GF nor TFM file; only the LOG file will appear in the current directory. The LOG file reads:

```
This is METAFONT (mf386),
       Version 2.718 [4b]
(preloaded base=plain 95.11.10)
         13 SEP 1996 13:13
**\&plain foo.mf
(foo.mf
This is a trivial program. )
```

The message

```
This is a trivial program.
```

will appear also on the screen.

Now, let us consider a bit more realistic program named, say, REC.MF:

```
 beginchar(48,  % ASCII code of character
          2cm#, % width of character
          1cm#, % height of character
          0cm#  % depth of character
          );
 fill unitsquare xscaled 2cm yscaled 1cm;
endchar;
```

A moderate knowledge of a programming language (or even plain English) and a moment of thought is sufficient to find out what character will be generated: obviously, a rectangle of dimensions $2\,\mathrm{cm} \times 1\,\mathrm{cm}$. The meaning of the statements used in the above program will be explained later.

This time METAFONT should be invoked differently, since now the resolution of the output device, for which the bitmap is meant, is essential. In the following example

```
mf386 \&plain \mode=hplaser; input rec.mf
```

the formula mode=hplaser is responsible for setting the resolution.[2] More precisely, the variable, mode, receives the value of the symbol hplaser which is set to an appropriate value during the process of generating the base (plain); the value of mode is used by the macro mode_setup, which tells METAFONT that a bitmap for a Hewlett-Packard

[1] Throughout the booklet, Eberhard Mattes's implementation of METAFONT for MS DOS, mf386, is referred to in examples.

[2] The backslash \ preceding the formula causes METAFONT to change the mode of the interpretation of command-line parameters: starting at the backslash, METAFONT expects to encounter statements written in its own lingo.

laser printer of resolution 300 × 300 pixels per inch is to be generated.

Three files will be created this time: REC.300GF, REC.TFM, and REC.LOG. REC.300GF (PC DOS abbreviates its name to REC.300) contains the description of the bitmap; REC.TFM contains information that the font consists of one character of code ASCII 48 and that the dimensions of the character are 2 cm × 1 cm; and REC.LOG contains text which is a little more elaborate than the previous example:

```
This is METAFONT (mf386),
      Version 2.718 [4b]
(preloaded base=plain 95.11.10)
         13 SEP 1996 13:13
**\&plain \mode=hplaser; input rec.mf
(rec.mf [48] )
Font metrics written on rec.tfm.
Output written on rec.300gf
(1 character, 520 bytes).
```

If the TEX installation at our site is equipped with drivers which can read GF files, the character generated a moment ago can now be printed. In order to do this, one should place the file REC.TFM in a directory searched by TEX, and, moreover, the file REC.300GF in a directory searched by the driver. The respective TEX program might look as follows:

```
\font\f rec\f0\end
```

If the installed TEX drivers do not accept GF files, they are bound to accept PK files, hence it suffices to convert the file REC.300GF to REC.PK. This can be done with the help of the program GFTOPK, belonging to the standard TEX distribution:

```
gftopk rec.300 c:\pxl\300\rec.pk
```

(c:\pxl\300\ is a hypothetical name of the directory, where the PK files of resolution 300 × 300 pixels per inch are to be collected.) The conversion GF → PK is always advantageous, as the PK files are more efficiently compressed.

It should be stressed that the presented method of generating the font REC is fairly universal. In particular, the fonts of the Computer Modern family can be generated using exactly the same scheme. The somewhat troublesome operation of copying the resulting files to appropriate directories need not be performed manually. In Eberhard Mattes's package, emTeX, one can find the program named MFJOB which neatly performs this part of job. In fact, MFJOB is devised to control the overall process of font production.

METAFONT as a Programming Language

Now that we are warmed up, let's look at some elements of the METAFONT lingo. Hopefully, the chosen subset is representative — I believe that the presented fragment will enable the reader to imagine the omitted part of the language. I apologize in advance for a virtual vagueness I am unable to avoid.

Variables

The notion of a METAFONT variable is somewhat peculiar. For the purpose of this article, however, it is enough to know that variables can be used in much the same way as in Pascal or C, except that the set of admissible characters is broader: besides letters (capital and small letters are distinguished) and digits, the name of a variable can contain such characters as a hash, an apostrophe, an exclamation mark, a question mark, a dollar sign, a tilde, etc.

The declaration of variables is not obligatory. Using an undeclared variable makes METAFONT interpret it as a variable of the numeric type (see the section "Numbers"). All variables are assigned initially a value "undefined", which is *not the same* as "not defined". This feature distinguishes METAFONT from other computer languages which either do not initialise variables by default (C, Pascal) or assign them a null value (AWK). It can be assumed that every programmer hunted fiercely at least once for a variable which was not assigned an initial value — it is really a tremendous task. From this point of view, METAFONT is safe, since a programmer can check from within a program (see the section "Logical values") whether a variable is initialised or not. As we shall see soon, it is not the only advantage of having this seemingly exotic possibility.

Units

Now, the time is ripe to explain a strange dualism of the units occurring in the program REC.MF: both cm# and cm appear. At a glance one may consider it to be a bug, but it is not a bug. On the contrary, the dualism is an important feature of METAFONT programs, and in order to understand them one should be aware of the source and the consequences of the dualism.

Well, the units followed immediately by a # denote quantities independent of resolution, so-called *sharp units*. In fact, they are variables which are assigned values in the plain format: pt#=1, cm#=28.45276, `mm#`=2.84528, `dd#`=1.07001, `bp#`=1.00375, `pc#`=12, `cc#`=12.84010, and `in#`=72.27. The user is expected not to change them. Their change — METAFONT does not prevent this — may likely cause havoc.

Observe that the unit values are expressed in points, hence a more appropriate name would be *point units*. In agreement with tradition, however, I will let the name *sharp units* stand.

All *sharp units* have their "hashless" counterparts, *pixel units*: `pt`, `cm`, `mm`, `dd`, `bp`, `pc`, `cc`, and `in`. Similar to the *sharp units*, they are numeric variables. They express the number of pixels falling into the interval of a respective length. In our example (recall that the resolution was set to 300×300 pixels per inch) the values of the pixel units are: `pt`=4.1511, `cm`=118.11055, `mm`=11.81102, `dd`=4.4417, `bp`=4.16667, `pc`=49.81314, `cc`=53.30035, and `in`=300. They are computed when executing `mode_setup`.

The metric files, i.e., the `TFM` files, contain sharp values, whereas pixel units should be used for drawing curves and filling areas. This simple trick facilitates the construction of METAFONT programs (provided some discipline is obeyed), since the programs become, in fact, resolution-independent. The following program:

```
1. mode_setup;
2. beginchar(48, 56.90552, 28.45276, 0);
3. fill unitsquare
4.  xscaled 236.2211 yscaled 118.11055;
5. endchar;
6. end
```

is, in principle, equivalent to the program `REC.MF`, but it has at least two drawbacks: first, it is significantly less intelligible; secondly, if a change of resolution is required, the fourth line of the program should be changed which would necessitate computing the respective values manually — who wants to do that?

Assigning values to variables

A METAFONT variable can be assigned a value in one of two ways: either by the use of an assignment symbol `:=` (as in `Pascal`), or by the use of an equality symbol, e.g., `2+x=3*y`. One can easily see that the two ways are not equivalent, as the statement `2+x:=3*y` appearing in a `Pascal` program would yield a translation error.

The former assignment method is common to all programming languages (only the assignment symbol may vary from language to language), and therefore it does not require thorough explanation. On the other hand, the latter method is so important, especially in the context of numerical calculations, that we shall dwell a bit longer on this subject.

In many cases, it is more natural and convenient to describe a graphic object, or a part of it, in terms of certain relationships rather than in terms of specific values, resulting from these relationships. If the relationships lead to an algebraic system of linear equations, METAFONT is well-suited to deal with such tasks — there is no necessity of solving the system by hand. The problem of finding an intersection point of two straight lines may serve as a characteristic example of such a task. The solution involves both METAFONT-specific equations as well as expressions with undefined values, as we shall see in the section "Vectors".

Data Types and the Relevant Operations

Logical (also called Boolean) values. METAFONT, like most programming languages, provides two logical constants: `true` and `false`. The logical expressions can be formed with relational symbols (`<`, `=`, `>`, etc.), logical operators (`and`, `or`, `not`, etc.) and braces, e.g., `(1<0) or (true<false)`. The logical expressions appear primarily in conditional and loop statements (see sections "Conditional statements" and "Iterative statements").

A logical variable can be declared using the instruction `boolean`; e.g., `boolean a,b,c` means that the names `a`, `b`, and `c` represent the logical variables from this point.

As has already been mentioned, METAFONT can check whether a given expression has a definite value. For this purpose the operators `known` and `unknown` can be used syntactically preceding the expression. The value of the expression `known b` immediately after the declaration of `b` is, obviously, `false`. (What is the value of the expression `known (known x)`?)

Furthermore, METAFONT provides for checking the type of an expression. In particular, the expression should be preceded with the operator `boolean` in order to check whether a given expression is of the logical type, e.g., `boolean(true and false)`. In general, the same operator can be used both for declaring a variable and for checking the type of an expression.

Strings. A sequence of characters not exceeding a single line, surrounded by a pair of double quotes denotes a string (text), e.g., `"this is a string aka text"`. Strings are mainly used for communication between the program and the surrounding world — we have already seen one example. Another place where one-character strings may appear is the statement `ligtable` (see section “Statement `ligtable`”).

Obviously, the instruction `string` is meant for declaring string variables.

Numbers. METAFONT, unlike typical programming languages, does not distinguish between integer and real (floating point) numbers. All variables that can take numeric values are uniformly declared by the instruction `numeric`. The fraction part of a number is always separated by a period (recall that in TeX both a period and a comma are admissible), and the integer part of a real number can be omitted, e.g.: `1234`, `.61804`, `3.14159`, etc.

METAFONT accepts typical expressions like `1+x`, `abs(x)`, `x*y/2`, `x-round(x)`, etc. Even more, it allows for omitting a times operator between a number and an expression, e.g., instead of `2*(x+3*y)` one can use a shorter form `2(x+3y)`, which is very convenient in practice.

Rational fractions are treated as numbers, i.e., the expression `1/2x` will be interpreted by METAFONT as $\frac{1}{2}x$, not as $\frac{1}{2x}$.

METAFONT offers a set of geometry-oriented algebraic operations. Among others, Pythagorean addition and subtraction, $\sqrt{x^2+y^2}$ and $\sqrt{x^2-y^2}$, are available. These operations, very useful in the process of creating graphic objects, are denoted by `++` and `+-+`, respectively. They represent binary (infix) operators, i.e., one uses them in expressions like `x++y` or `x+-+y`.

Yet another METAFONT-specific operation is *mediation* between points (“of-the-way” function), especially useful in the context of expressing relations between points on a plane. Instead of saying `(1-t)*x+t*y`, one can simply say `t[x,y]`, where `t` can be an arbitrary expression. In particular, `t` can be a variable.

With this notation, METAFONT differs from `AWK` or `Pascal`. Typically, `x[2,5]` denotes a variable with two indices, $x_{2,5}$ in mathematical notation. For METAFONT, it is a linear expression yielding 2 for x=0 and 5 for x=1; in other words, the formula `x[2,5]` is equivalent to `3x+2`. The variable with two subscripts should be represented in METAFONT as `x[2][5]`, which is also a convention accepted by `Pascal`. There is virtually no limit for the number of indices in METAFONT.

A somewhat peculiar numeric quantity, `whatever`, is predefined in `plain`. Formally, it is a parameterless function yielding a numeric undefined value. The question arises: what is such a fancy constant for? The answer will soon emerge...

Vectors. As a program designed to operate on a plane, METAFONT is equipped with pairs of numbers that can be interpreted as points or vectors of a Cartesian plane. The notation is intuitive and simple: given two numeric values (expressions) `x` and `y`, the formula `(x,y)` represents the expression of type “pair”. The instruction `pair` can be used for declaring pair variables and for checking the type of expressions.

There are two functions, specific for this data type, taking a pair expression as an argument and returning a numeric value, namely, `xpart` and `ypart`. Their meaning is obvious: `xpart((x,y))` = `x`, `ypart((x,y))` = `y`.

Five useful vectors are predefined in `plain`: `origin` $=(0,0)$, `right` $=(1,0)$, `left` $=(-1,0)$, `up` $=(0,1)$, and `down` $=(0,-1)$.

Pairs of the form `(x`$\langle$*anything*$\rangle$`,y`$\langle$*anything*$\rangle$`)`, where $\langle$*anything*$\rangle$ denotes a valid ending part of a name (suffix), can be abbreviated to `z`$\langle$*anything*$\rangle$; e.g., one can write `z123` or `z'` instead of writing `(x123,y123)` or `(x',y')`, respectively. In particular, formulas `(x,y)` and `z` are equivalent, provided `x` and `y` are numeric variables, which is usually the case, unless a mad user defines them otherwise. It should be noted that it is not a built-in convention — the notation is due to a smart definition of the symbol `z`.

The operation of mediation, described in the section “Numbers”, can also be applied to vectors. In this case — the interpretation is self-suggesting — `t[z1,z2]` denotes a point, belonging to the segment with endpoints `z1` and `z2`, such that the segment is divided by this point in the proportion $t:(1-t)$. For example, `1/2[z1,z2]` denotes the midpoint of the segment, `0[z1,z2]` denotes the point `z1`, `1[z1,z2]` denotes the point `z2`.

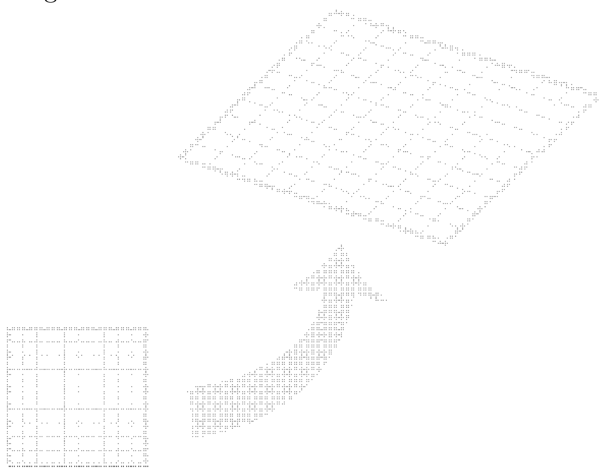
A truly useful paradigm of METAFONT programming can now be demonstrated: *given points* `z1`*,* `z2`*,* `z3`*, and* `z4` *such that the line determined by* `z1`*,* `z2` *and the line determined by* `z3`*,* `z4` *are not parallel, find the point where the two lines cross.* The natural METAFONT solution is elegant, although perhaps somewhat surprising:

```
z5=whatever[z1,z2]=whatever[z3,z4]
```

Indeed, it is the demanded solution, since — according to what has already been said — `whatever[z1,z2]` denotes *a certain* point belonging to the line drawn through the points `z1` and `z2`; similarly, `whatever[z3,z4]` denotes *a certain* point belonging to the line drawn through the points the z3 and z4. Therefore, the point z5 belongs to both lines.

Actually, the foregoing formula is interpreted by METAFONT as a system of linear equations which, under the stated assumptions, has a unique solution, z5.

Affine transformations. Besides the pairs of numbers, METAFONT provides also 6-tuples, representing affine (linear) transformations of a plane. Affine transformations convert squares into parallelograms:



A METAFONT user need not be initiated into the mysteries of mathematics in order to use transformations efficiently. Operations with self-explanatory names, such as `shifted`, `rotated`, `slanted`, `xscaled`, `yscaled`, and similar names, suffice in most cases.

The following objects can be subject to affine transformations: vectors, paths, pens (see below) and, of course, transformations.

For example, if a path `p` is to be translated horizontally by 2 cm, the following construction can be used:

```
p shifted (2cm,0)
```

Similarly,

```
z0 rotated 55
```

denotes the counter-clockwise rotatation of the vector `z0` by 55 degrees (a positive angle denotes a counter-clockwise rotation);

```
p xscaled 2 yscaled 2
```

denotes the magnification of the path `p` by factor 2 (in this case, a simpler form can be applied: `p scaled 2`);

```
p reflectedabout (z1,z2)
```

denotes the mirror symmetric image of the path `p` about the line drawn through the points `z1` and `z2`; and so on.

The user can declare transform variables using the instruction `transform`. In order to use such a variable, the following construction can be used: ⟨*object*⟩ `transformed` ⟨*transformation*⟩, e.g., `z0 transformed A`, where `A` is a variable of type `transform`.

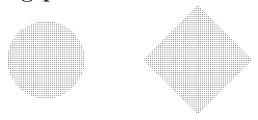
METAFONT also provides access to all numeric components of a transformation, namely, there are six functions `xxpart`, `xypart`, `yxpart`, `yypart`, `xpart`, and `ypart` which for a given transformation yields the respective components. A less experienced user need not bother about transform variables and their components — they appear comparatively seldom in applications.

Pens. We now know almost enough to draw a simple picture, except for one METAFONT tool — pens. Let's pass immediately to an example without going into theoretical details:

```
1. pickup pencircle scaled 1cm;
2. draw (0,0);
3. pickup pensquare scaled 1cm rotated 45;
4. draw (2cm,0);
```

Typical parts of a METAFONT program, such as `mode_setup`, `beginchar`, etc., have been omitted, as they are unimportant here.

The first line contains the instruction `pickup pencircle` which tells METAFONT use a circular pen, 1 cm in diameter; the second line tells METAFONT to use the currently chosen pen to draw a "dot" in the origin of the coordinate system. Similarly, the final two lines instruct METAFONT to put a "square dot" at the point (2 cm, 0). The resulting figure is admittedly trivial, nonetheless, it is a good starting point:



Paths. It is nearly impossible to imagine a graphic system without objects corresponding to planar curves. Obviously, METAFONT provides objects of this kind, called paths. They are declared using the instruction `path`. Each path consists of segments being third-order arcs, known as Bézier curves. Such a segment is determined uniquely by four points z_0, z'_0, z'_1, and z_1 (z'_0 and z'_1 are called *control points*); for $t \in \langle 0, 1\rangle$ the intermediate points of a Bézier curve are given by the following formula:

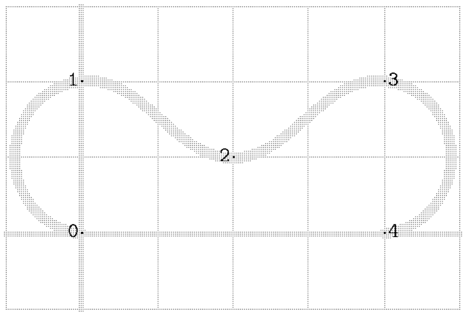
$$z_0\,(1-t)^3 + 3z'_0\,t(1-t)^2 + 3z'_1\,t^2(1-t) + z_1\,t^3$$

As in the case of affine transformations, a budding METAFONT user can ignore all intricate subtleties of mathematics connected with Bézier curves. It is the simplicity of the foregoing formula that is important here. Worth mentioning is also the parametrization of Bézier curves (the parameter t is sometime referred to as "time"): as t increases from 0 to 1, the formula yields coordinates, in order, of all points belonging to the curve. For $t = 0$ and $t = 1$, the formula returns the coordinates of the edges, z_0 and z_1, respectively. Some of METAFONT path operations, e.g., the operation `subpath`, refer to the parameter t (see below).

One of the most striking capabilities of METAFONT is its skill at interpolating.[3] The excellent and efficient interpolation mechanism is undoubtedly one of the best features of METAFONT. To see how it works, let's assume that a curve is to be drawn through the points `z0` $= (0, 0)$, `z1` $= (0, 2\,\mathrm{cm})$, `z2` $= (2\,\mathrm{cm}, 1\,\mathrm{cm})$, `z3` $= (4\,\mathrm{cm}, 2\,\mathrm{cm})$, and `z4` $= (4\,\mathrm{cm}, 0)$. If no additional constraints are imposed, such a task can be expressed in METAFONT as follows:

```
draw z0..z1..z2..z3..z4
```

The operation "horizontal colon" causes METAFONT to employ its interpolation methods, trying to join Bézier arcs as smoothly as possible. The result you can see in the following figure (the grid was added in order to facilitate the readings of the coordinates of nodes).

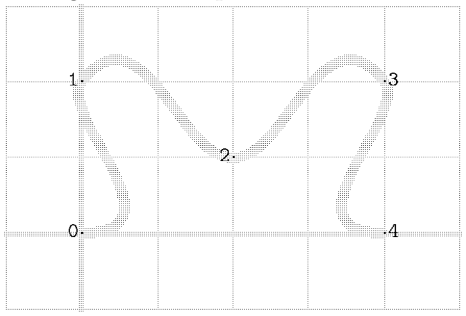


According to my experience, I would suggest that the designers of commercial graphic systems consult the source code of METAFONT in order to improve the interpolation involved in their systems.

The process of interpolation can be controlled by imposing constraints. One such constraint is to force the direction at a given node. To do this, an appropriate vector should be added in curly braces at chosen nodes in a path formula, e.g.:

```
draw z0{right}..z1..z2..z3..{right}z4
```

(recall that `right` denotes the vector `(1,0)`). The local change of constraints causes seemingly the global change of the shape of the curve:

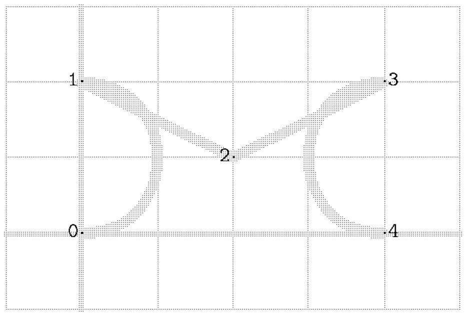


In fact, the disturbance is nearly local. More precisely, it vanishes exponentially when going away from the point of change. If the curve consisted of a greater numbers of nodes, the effects of the disturbance would be imperceptible only a few nodes away from its source.

Besides the "horizontal colon", there are also other path operations. Frequently, the "double-dash" operator, representing a straight-line connection, is used. For example, the formula

```
draw z0{right}..z1--z2--z3..{right}z4
```

results in



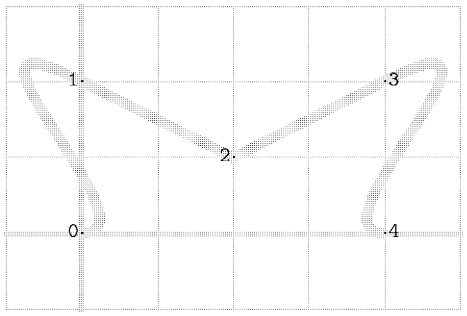
The "double-dash" operator causes the neighbouring segments to be calculated independently as if they were disconnected. The control points of straight-line segments defined in such a way fulfill the relation $z'_0 = \frac{1}{3}[z_0, z_1]$, $z'_1 = \frac{2}{3}[z_0, z_1]$; in other words, the control points and the endpoints are equidistant.

If a smooth connection of straight lines and arcs is required, the "triple-dash" operator can be used:

```
draw z0{right}..z1---z2---z3..{right}z4
```

which yields the following change of the curve:

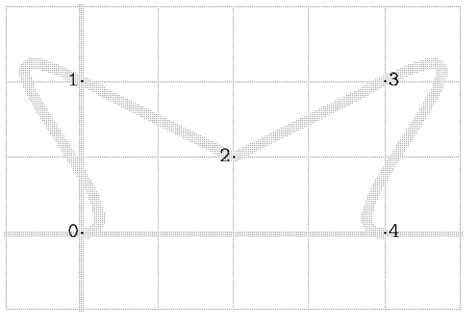
[3] METAFONT's interpolation machinery was worked out by John D. Hobby and was published in his thesis at Stanford University. His idea was to keep the overall curvature of the resulting curve constant, if possible. It turns out that the human eye is extremely sensitive to the changes of curvature, hence the human inclination to perceive curves with smoothly changing curvatures as aestethically pleasing.



This method, however, has one drawback. Namely, the control points of segments marked by the triple dash almost coincide with the edges of the segments. METAFONT does not see anything particular in such a singularity. If, however, exporting to other systems is intended, the usage of the triple dash should be discouraged, unless the user is aware of what is being done. A safer method is to supply the direction at the nodes explicitly and to apply the double dash; in such a case the respective path formula would take the form:

```
z0{right}..{z2-z1}z1--z2--z3{z3-z2}..
            {right}z4
```

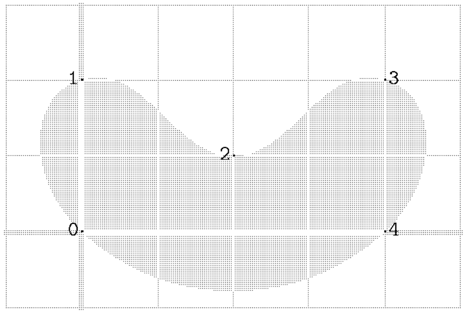
The Bézier straight-line segments are "tidy" and the shape of the curve stays almost intact. (Check it.)



The paths considered so far did not form a closed contour. In order to convert an open curve into a closed contour, the path should be ended by the operation `cycle`. Closed contours are important as they can not only be drawn but also can be darkened with the operator `fill`:

```
fill z0..z1..z2..z3..z4..cycle
```

The resulting figure is displayed below:



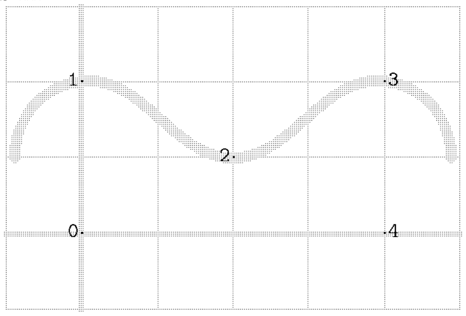
More about paths. A reverse operation to joining segments is, in a sense, an operation that pulls a fragment out of a path. This can be accomplished in METAFONT by the use of the operator `subpath`. The previously mentioned notion of the parametrization of Bézier curves is crucial here. The notion was formulated for single segments. Its generalization for multisegment paths is straightforward: nodes are numbered from 0 upwards. As the parameter t takes on (real) values from $i-1$ through i, the corresponding point traverses the path from the node $i-1$ to the node i. Assume that two numbers, u and v, are given; the fragment of a path p corresponding to the interval (u,v) can be expressed in METAFONT lingo as

```
subpath (u,v) of p
```

Referring to our previous example, the statement

```
draw subpath (.5,3.5) of
     (z0..z1..z2..z3..z4)
```

results in

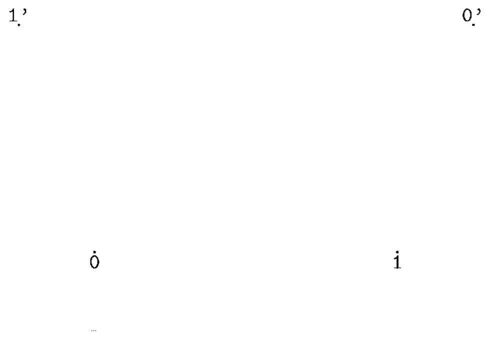


The operation `subpath` always produces non-cyclic paths, even if the operand forms a closed contour.

Although the path operations we have seen so far suffice for most applications, there exists a general path construction, enabling a fastidious user to shape curves arbitrarily:

```
draw z0 .. controls z0' and z1' .. z1
```

The construction `z0 .. controls z0' and z1' .. z1` corresponds precisely to the formula given at the beginning of the section "Paths". For example, the figure



can be generated by the following short program

```
z0=(0,0);
z0'=(5cm,3cm);
z1'=(-1cm,3cm);
z1=(1cm,3cm);
draw z0 .. controls z0' and z1' .. z1
```

A few handy paths have been predefined in the `plain` format. Two of them are particularly useful: `unitsquare`, i.e., a square of the side length equal to 1 and the lower left corner coinciding with the origin of the coordinate system (cf. example `REC.MF`) and `fullcircle`, i.e., a circle whose diameter is equal to 1 and whose centre lies at the origin of the coordinate system. Both are, obviously, cyclic paths.

Supplementary path operations. Furthermore, there are a few path operations characterizing a point on a path. Two of them, `point` and `direction`, are most frequently used. The operation `point` yields coordinates of the point of a curve corresponding to the value of a given parameter `t`. The operation `direction` returns a vector parallel to the direction of the path at a point corresponding to a given time `t`. A sample code illustrating the usage of these operations is given below:

```
z0=point t of p;
z1=z0
  +1mm*(unitvector(direction t of p)
        rotated 90);
z2=z0
  +1mm*(unitvector(direction t of p)
           rotated -90);
```

Point `z0` lies, obviously, on the path `p`; pont `z1` lies 1 mm to the left (with respect to the path direction) of point `z0`; and point `z2` lies 1 mm to the right of point `z0`.

There is also a dual operation to `direction`, namely, the operation `directiontime`. It returns a real number `t` such that for a given vector `d` and a given path `p` the equality `direction t of p = d` holds. For example, the value of the expression

```
directiontime up of ((0,0){right}..
                     {left}(0,1))
```

is 0.5, which could easily be guessed.

We have already dealt with the problem of finding a common point of two straight lines. METAFONT is prepared for performing a more general task. Namely, there exists an operation `intersectiontimes` which finds a crossing point for two arbitrary paths. Assume that two paths, `p1` and `p2`, are given. The equation

```
(t1,t2) = p1 intersectiontimes p2
```

defines two numbers, `t1` and `t2` such that

`point t1 of p1` $\approx$ `point t2 of p2`

(the approximate equality is unavoidable due to rounding errors).

If paths do not touch each other, the result of the operation `intersectiontimes` is $(-1,-1)$; if there are several points where they touch each other, the operation yields the first feasible point.

Arrays. Variables of all types can be declared as indexed arrays. In order to do this, the name declared should be followed by one or more pairs of square brackets, e.g.,

```
transform T[][]; pair d[];
```

Now, you can say `T[i+j][k]` (provided `i`, `j` and `k` are numeric), `d[0]`, or even `d[1.5]`, as METAFONT allows for indexing with real numbers (they are not rounded), etc. If the index expression is a number only, the square brackets can be omitted, i.e., `d[0]` is equivalent to `d0`, `T[1][2]` is equivalent to `T1 2`, `z[0]'` is equivalent to `z0'`, and so on. This convention is METAFONT-specific.

Numeric arrays need not be declared. The first occurrence of a variable, say, `q0` causes an implicit declaration `numeric q[]`.

At last, the description of data types and the related operations has come to an end. We are a few paces from sensible applications. One important subject, however, has not been treated yet — statements.

Statements

We have already seen a lot of statements, e.g., `message`, `fill`, `draw`, `beginchar`, `endchar`, `mode_setup`, to mention some of them. The program can be built out of such primary statements in three ways: (1) statements can be executed sequentially, one after the other — to mark this a semicolon is used; (2) one among several statements can be performed, provided a certain condition holds — these are conditional statements, or conditions; (3) a given statement can be repeated as long as a certain condition holds — these are iterative statements, or loops.

First, some primary statements will be described, followed by conditional and iterative statements, and then we will deal with a more elaborate example.

The statements `beginchar` and `endchar`. Both statements have already appeared (see example `REC.MF`). Needless to say, statements of this kind should be present in any language devised for rendering fonts. The details of their behaviour are somewhat complex, but fortunately, we can slide over this subject, as from the practical point of view they are not essential.

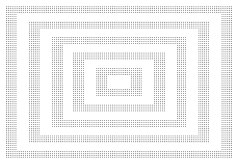
The statement `beginchar` assigns values to METAFONT's internal variables `charcode`, `charwd`, `charht`, and `chardp` according to the values passed as parameters to the statement (four comma-separated numbers enclosed by braces). They refer to the ASCII code and to the width, height, and depth of the character, respectively. The dimensions should be given in *sharp units*. Furthermore, the variables `w`, `h`, and `d` receive the values corresponding to `charwd`, `charht`, and `chardp`, but expressed in pixel units. When programming characters, these variables come in handy.

The parameterless instruction `endchar` ends the code for a given character. Once METAFONT reads this statement, the values of `charcode`, `charwd`, `charht`, and `chardp` are written out to the `TFM` file, and the bitmap of the character is written to the `GF` file. Next, variables such as `x`, `y` (and hence `z`; see section "Vectors") `w`, `h`, and `d` are initialised, therefore the user need not bother about the values assigned previously when dealing with subsequent characters.

Both `beginchar` and `endchar` are defined in the `plain` format, thus a fastidious user can adjust them to meet particular needs.

The statements `fill`, `draw`, and `erase`. So far, we have become familiar with the statements `fill` and `draw`; the operator `erase` prepended to any of them causes painting in white rather than in black.

The following example demonstrates the results of the usage of the operations `fill` and `erase fill`:



The above figure was obtained by the following program:

```
mode_setup;
beginchar("0",3cm#,2cm#,0);
 pair c; c=(.5w,.5h); % centre of
                      % the character
 path q; % a unit square with a centre
         % coinciding with the origin
         % of the coordinate system
 q=unitsquare shifted (-.5,-.5);
 fill q xscaled w
     yscaled h shifted c;
 erase fill q xscaled .9w
     yscaled .9h shifted c;
 fill q xscaled .8w
     yscaled .8h shifted c;
 erase fill q xscaled .7w
     yscaled .7h shifted c;
 fill q xscaled .6w
     yscaled .6h shifted c;
 erase fill q xscaled .5w
     yscaled .5h shifted c;
 fill q xscaled .4w
     yscaled .4h shifted c;
 erase fill q xscaled .3w
     yscaled .3h shifted c;
 fill q xscaled .2w
     yscaled .2h shifted c;
 erase fill q xscaled .1w
     yscaled .1h shifted c;
endchar;
end
```

Actually, it is a "naive" version of the program. An improved version appears in the section entitled "Iterative statements".

The statement `ligtable`. This statement has more to do with a font as a whole rather than with the shapes of individual characters. The general form of the statement `ligtable` is by far too complex to be described here entirely — we shall confine ourselves to the definition of *kerns*. Kerns are tiny spaces, possibly negative, inserted when the room between a pair of characters is optically either too small or (more frequently) too large. Kerns defined by the statement `ligtable` are presumably known to the TEX user as *implicit kerns*. The information about implicit kerns is written to a `TFM` file at the end of METAFONT's run.

It should be emphasized that kerns are vital for the final appearance of the font. Improper kerning can spoil a font even if the character shapes are masterfully designed.

A typical example of a word in which kerns are required is the word "WAY". The letters in both pairs, "WA" and "AY", would be too far from each other without kerning:

rather this WAY *than this* W A Y

Here you have an excerpt from the `ligtable` program for the font `CMR10`.

```
k#:=-5/18pt#; kk#:=-5/6pt#;
              kkk#:=-10/9pt#;
ligtable "F": "V": "W":
       "o" kern kk#, "e" kern kk#,
       "u" kern kk#, "r" kern kk#,
        "a" kern kk#, "A" kern kkk#,
    "K": "X":
        "O" kern k#, "C" kern k#,
        "G" kern k#, "Q" kern k#;
ligtable "A": "R":
        "t" kern k#, "C" kern k#,
        "O" kern k#, "G" kern k#,
        "U" kern k#, "Q" kern k#,
    "L":
        "T" kern kk#, "Y" kern kk#,
        "V" kern kkk#, "W" kern kkk#;
```

The first two lines of the excerpt defines three degrees of kerning to be used subsequently. One-letter strings followed by a colon refer to the left-hand sides of kern pairs, whereas one-letter strings followed by the operator `kern` refer to the right-hand sides of kern pairs. The right-hand sides are to be paired with all preceding left-hand sides. Such a notation allows for specifying a great number of kern pairs in a compact and legible way, e.g., the first `ligtable` statement specifies 38 kern pairs. (Why? How many kern pairs specifies the second `ligtable` statement?) The kerns under consideration read `kkk#` for "WA" and `kk#` "AY". (Check it in TeX.)

It is the information produced by `ligtable` statements that is responsible for the size of `TFM` files, hence the kern pairs that are unlikely to occur, e.g., "yY", should be avoided. Incidentally, the pairs "Av" and "Aw" are absent from the kern pairs of the Computer Modern family, which I am inclined to consider a drawback.

Finally, let's quote Donald E. Knuth's admonition concerning the adjustment of the amount of kerning:

> *Novices often go overboard on kerning. Things usually work out best if you kern by at most half of what looks right to you at first, since kerning should not be noticeable by its presence (only by its absence). Kerning that looks right in a logo or in a headline display often interrupts the rhythm of reading when it appears in ordinary textual material.*
>
> The METAFONT book, p. 317

The statements `end` and `bye`. These statements, similar to TeX's `\end` and `\bye`, trigger last-minute actions. Among others, the information about kerns is being written to the `TFM` file. Afterwards, METAFONT closes the process of data processing. As in TeX, both statements can be thought of as synonyms.

Conditional statements. The simplest conditional statement has the following form:

`if` ⟨*logical expression*⟩ `:` ⟨*statement*⟩ `fi`

which means that ⟨*statement*⟩ is to be executed if and only if ⟨*logical expression*⟩ takes on the value `true`. The symbol ⟨*statement*⟩ stands not necessarily for a primary statement; it can be an arbitrarily complex construction, involving loops, conditions and their sequences.

A more general form, often indispensable, is:

`if` ⟨*logical expression*⟩`:` ⟨$statement_1$⟩
`else:` ⟨$statement_2$⟩ `fi`

In this case, ⟨$statement_1$⟩ is performed if ⟨*logical expression*⟩ holds, and ⟨$statement_2$⟩ otherwise.

The moral is that METAFONT's conditions differ mainly in syntax from those of `Pascal` or `C`, while the semantics are equally straightforward.

Iterative statements. The reason for using such statements has already appeared: in the example demonstrating the usage of the operation `erase`, a series of almost identical statements occurs, except that the numbers occurring in the statements vary. Iterative statements are suitable in such cases. METAFONT's `for` statement, syntactically similar to the statement `for` of `Algol 60` (who remembers it?), allows for the replacement of the lines 9–28 of the mentioned example by a more compact code:

```
 1. for i:=10 step -2 until 2:
 2.  fill q
 3.   xscaled (1/10i*w)
 4.          yscaled (1/10i*h)
 5.   shifted c;
 6.  erase fill q
 7.   xscaled (1/10(i-1)*w)
 8.          yscaled (1/10(i-1)*h)
 9.   shifted c;
10. endfor
```

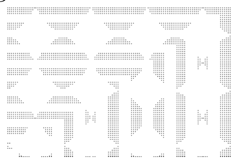
The meaning of the code can be explained as follows: `i` is a local variable which takes on values starting from 10 with step −2 until the value 2 is reached, i.e., the "looped" statement (lines 2–9) is performed for `i`=10, `i`=8, `i`=6, `i`=4, and `i`=2.

The code can be compacted further by using a conditional statement:

```
for i:=10 downto 1:
 if odd i: erase fi fill q
  xscaled (1/10i*w) yscaled (1/10i*h)
  shifted c;
endfor
```

The operation `downto` is equivalent to `step -1 until`; the expression `odd i` yields `true` if `i` is an odd number and `false` otherwise.

Loops are useful not only as a means of abbreviating programs; first of all, they enhance the expressive power of a language and thus facilitate the modifications of programs. In order to obtain the following figure



a simple cosmetic change of the recent version of the program is needed:

```
for i:=20 downto 1:
  if odd i: erase fi fill q
   xscaled (1/20i*w) yscaled (1/20i*h)
   shifted (1/20i*c);
 endfor
```

Imagine how long the code would be without a loop and how laborious the respective change would be.

The description of conditional and iterative statements is far from being complete. Our knowledge, however, is sufficient to understand the examples I am about to demonstrate.

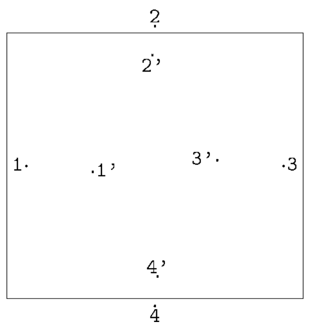
Examples

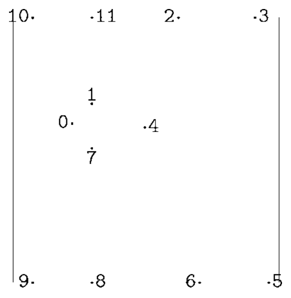
The title of this article suggests that the first example should bear a stamp of practicality. Needless to say, the truly practical applications are infested with obscure details. Therefore the following example, the font **OK**, should be regarded as a model of reality rather than reality itself.

Font OK. The font **OK** contains only two letters, namely, **K** and **O**. The font is admittedly simple. This does not mean that it cannot serve as an ample example. On the contrary, it turns out that the detailed description of this simple font is surprisingly long. It is by no means a drawback of METAFONT — just that the task of font design is intrinsically difficult. The complexity of METAFONT programs is a derivative of the complexity of the task.

The font **OK**, like the fonts of the Computer Modern family, consists of a parameter file (primary), `OK10.MF`, and a driver file (secondary, to be input), `OK.MF`. The parameter file defines a set of numeric quantities, specific for a given nominal size (10 pt), whereas the driver file defines in a generic way the shapes of characters.

The magnified letters **O** and **K** of the font **OK** are shown below:





Notice the nodes marked with `0`, `1`, `1'`, etc. They correspond to the variables `z0`, `z1`, `z1'`, ..., `z11`, respectively. To show them in action, both programs are presented *in extenso*. The reader is supposed to decide which parts of the code are worth reading and which can be skipped.

Let's peep at the file `OK10.MF`:

```
 1. s#:=10pt#;   % nominal font size
 2. u#:=1/18s#;  % unit width
 3. h#:=3/4s#;   % height of letters
 4. marg#:=u#;   % sidebar size
 5. o#:=1/50s#;  % top and bottom overshoot
 6.              % of the letter ``O''
 7. alpha:=5;    % angle of the torsion
 8.              % of the inner and outer
 9.              % edges of the letter ``O''
10. stem#:=3u#;  % thicknes of the arm
11.              % of the letter ,,K''
12. input ok
```

The first two lines are presumably obvious. Doubts may arise at the third line: why does the height of letters differ from the font size? There is no rule for that. Usually the size of a font is roughly the same as the overall height of a brace. Although the font **OK** does not contain a brace, it was intended to be used with the font `CMR10` in which letters are roughly 7 points tall.

Line 4 defines the distance between the glyph of a character and the side edges of a character. The width of a character is usually a bit greater than the width of its glyph. In the case under consideration, the letters would touch each other in the word **OK** if the variable `marg#` was assigned a null value. Note that, in general, left and right sidebars need not be equal.

Line 5 sets the amount of a so-called *overshoot.* This quantity is necessary for achieving the optical balance between the heights of rounded and square letters. The reason behind this is a well known optical illusion. Namely, a square and a circle of the same height are not perceived as being equal, a circle is seemingly smaller:

How to compensate for this illusion? Don't expect it to be a trivial task. An expert in the realm of computer fonts, Peter Karow (URW), says:

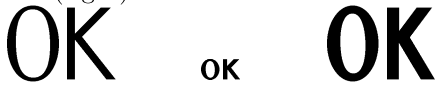
> *These and other optical effects can only be properly and correctly considered by experienced type designers. In future all technicians should bear this fact in mind. Let us hope that we have seen the last of those "computer typefaces in 3 hours."*
>
> Digital Formats for Typefaces, p. 26

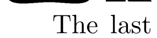
Line 7 defines the asymmetry of the inner and outer contours of the letter **O**. It is the matter of a designer's taste whether such an asymmetry is at all needed. In the font **OK** the value of 5 degrees has been arbitrarily assumed, but there are no profound reasons to stick to this value.

Eventually, the thickness of the arms of the letter **K** is determined in line 10.

Altogether, there are seven parameters—pretty few in comparison with the sixty two parameters of the Computer Modern family. But, on the other hand, surprisingly many for such a nearly trivial example.

The parameters allow for generating a broad variety of alterations. In particular, the font designer can obtain effects which cannot be achieved by simple non-uniform scaling. Let's set, e.g., `u#:=1/24s#` and `stem#:=2u#`. Compare the resulting light narrow font (left) with the original one (centre) and with the original font narrowed by factor 0.75 (right):



A careless change of parameters may lead to surprising and/or unwanted results, e.g., setting `u#:=1/4s#` causes a hardly acceptable effect: .

The last line of the file `OK10.MF` contains the statement `input ok`. METAFONT's `input` statement works essentially in the same way as TEX's `\input` statement: after reading it, METAFONT switches to the file `OK.MF` and continues to interpret the program. Following METAFONT, let's also switch to the file `OK.MF`. The METAFONT code becomes now somewhat tougher, therefore the reader is supposed to be armed with patience.

The two initial lines of the file read:

```
 1. mode_setup;
 2. define_pixels(stem,marg,o);
```

We are already acquainted with `mode_setup`. The statement `define_pixels` remains unknown thus far, but its meaning can easily be deduced. Actually, it assigns values to the implicitly declared variables `stem`, `marg`, and `o`. Obviously, the values are expressed in pixel units and correspond to the values of `stem#`, `marg#`, and `o#`, respectively.

The subsequent lines contain the description of the letter **O**:

```
 3. beginchar("O",15u#,h#,0);
 4.  z1=(marg,1/2h);
 5.  z1'=z1+9/8stem*
 6.       (right rotated -alpha);
 7.  z2=(1/2w,h+o);
 8.  z2'=z2+1/2stem*
 9.       (right rotated (-90-alpha));
10.  z3=(w-marg,1/2h);
11.  z3'=z3+9/8stem*
12.       (right rotated (180-alpha));
13.  z4=(1/2w,-o);
14.  z4'=z4+1/2stem*
15.       (right rotated (90-alpha));
16.  fill z1..z2..z3..z4..cycle;
17.  erase fill z1'..z2'..z3'..z4'..cycle;
18. endchar;
```

Note the intense usage of the variables `w` and `h` (cf. section "Statements `beginchar` and `endchar`"). Observe also that the first of the four parameters passed to `beginchar` is not a number. Instead, it is a one-letter string. METAFONT accepts such a variant, presuming that the ASCII code of the letter is meant, 79 in this case.

The next three lines prepare two auxiliary variables to be used in the program for the letter **K**.

```
19. pair K', K''; % vectors determining
20.               % the angle between the
21.               % arms of the letter `K'
22. K'=unitvector(1,1);
23. K''=unitvector(4/5,-1);
```

The operation `unitvector`, occurring in lines 22–23, computes a vector of length 1, parallel to the vector passed as an argument. Usually, it is more convenient to formulate relations without paying attention to the length of vectors (in this case `K'` $= (1/\sqrt{2}, 1/\sqrt{2})$, `K''` $= (4/\sqrt{41}, -5/\sqrt{41})$, admittedly ugly formulas, aren't they?), but in order to control distances between elements of a graphic object, unit-length vectors come in handy.

Now, a relatively complex program for the letter **K** ensues:

```
24. beginchar("K",0,h#,0);
25. % the width will be computed soon...
26.  forsuffixes $:= ,#:
27.   stem$'=11/12stem$;
28.   z0$=(marg$+2/3stem$,3/5h$);
29.   z1$=whatever[z0$,z0$+K'];
30.   x1$=marg$+stem$;
31.   z2$=whatever[z0$,z0$+K'];
32.   z3$+whatever*K'=z2$+stem$'*
33.         (K' rotated -90);
34.   y2$=y3$=h$;
35.   z7$=whatever[z0$,z0$+K''];
36.   x7$=marg$+stem$;
37.   z6$=whatever[z0$,z0$+K''];
38.   z5$+whatever*K''=z6$+stem$'*
39.         (K'' rotated 90);
40.   y5$=y6$=0;
41.  endfor
42.  charwd:=x5#+.5marg#;
43.  z4=whatever[z3,z3+K']=
44.         whatever[z5,z5+K''];
45.  z8=(marg+stem,0);
46.  z9=(marg,0);
47.  z10=(marg,h);
48.  z11=(marg+stem,h);
49.  fill for i:=1 upto 11:
50.         z[i]-- endfor cycle;
51. endchar;
```

The main source of the complexity is a peculiar principle underlying the construction of the letter: if the thickness and the directions of the arms are given, the width cannot be imposed, but has to be calculated. In this case, the width is controlled by the rightmost point of the letter **K**, i.e., by z_5. The width is set only in line 42. It is assigned a value of the x-coordinate of the point z_5 increased by the value of the variable `marg#` (cf. also sections "Vectors" and "Statements `beginchar` and `endchar`").

The tricky part is the loop in line 26. It works as follows: its body (lines 27–40) is performed twice; the control variable of the loop, `$`, is replaced by an empty suffix during the first pass, whereas during the second pass it is replaced by a hash. In other words, during the first pass the body will be interpreted as

```
stem'=11/12stem;
z0=(marg+2/3stem,3/5h);
z1=whatever[z0,z0+K'];
   ...
```

and during the second pass as

```
stem#'=11/12stem#;
z0#=(marg#+2/3stem#,3/5h#);
z1#=whatever[z0#,z0#+K'];
   ...
```

The second pass is necessary to compute the coordinates of z_5 in *sharp units*. Actually, the statement `mode_setup` defines the variable `hppp` (horizontal pixels per point), and one might try to compute `z5#` as equal to `z5/hppp`. This, however, is wrong, as the value of `z5#` would then depend on a given resolution due to rounding errors. The employed trick ensures that the `TFM` file is resolution-independent.

In order to understand the code in details, an unaided study is unavoidable. Therefore, we'll go no further into the matter, merely pointing out the characteristic features of the code.

The problem of finding a point where two straight lines cross (see section "Vectors") occurs several times here, hence the intense usage of equations and of the construction `whatever[...]`. Another interesting element is the loop in lines 49-50. It is used *inside* a path expression. It is a METAFONT-specific feature. Typical programming languages do not allow for using loops in expressions, while METAFONT accepts such constructions. For example, the statement

```
message decimal(for i:=1 upto 100:
                +i endfor)
```

will result in writing to the screen and to the `LOG` file the value 5050, i.e., the sum $\sum_{i=1}^{100} i$. Actually, the `for` loop can be thought of as a macro (TEX users are supposed to be familiar with the notion of macros), expanding in this case to `+1+2+3` ... `+100`, and that's the point. The operation `decimal` converts the numerical result to a decimal string representation, i.e., to `"5050"`.

The file `OK.MF` ends with the following sequence of statements:

```
52. ligtable "K" : "O" kern -3/2u#;
53. font_size s#;
54. font_slant 0;
55. font_normal_space 6u#;
56. font_normal_stretch 3u#;
57. font_normal_shrink 2u#;
58. font_quad 18u#; % 18u#=s#
59. bye
```

An extremely simple form of the statement `ligtable` appears in line 52. The first line defines one implicit kern to be inserted between **K** and **O**. The next six lines define six basic font parameters. Lines 54–58 can be accessed in TeX as `\fontdimen` registers, namely, `\fontdimen1`, `\fontdimen2`, `\fontdimen3`, `\fontdimen4`, and `\fontdimen6`, respectively (see *The TeXbook*, p. 433). The font size, also called *design size*, presents a little puzzle to TeX users: how to access a font size in a TeX program? (Hint: it is not `\fontdimen0`.) TeX makes use of the design size of a font when the font is declared using an `at` clause. For example, the statement

```
\font\f ok10 at 20pt
```

informs TeX that the font `OK10` should be loaded at doubled size, as the design size of the font is 10pt (see the first line of the file `OK10.MF`). A number appearing in a font name is traditionally equal to the design size of a font, but it is not advisable to rely on this information. In fact, TeX ignores it completely.

Our font in miniature is ready. The miniature, however, turned out to be fairly complex. I would consider my goal to be reached (at least partially), if the reader is not surprised to learn that the manual for the Computer Modern family is about six hundred pages long.

Solving systems of linear algebraic equations. In the handbooks of elementary algebra one can find exercises like this: *given a system of linear equations:*

$$a+b+c=1$$
$$a+2b+3c=1$$
$$3a+5b+9c=1$$

find numbers a, b, and c. It turns out that METAFONT is well-suited for solving algebraic problems of this kind. It just suffices to copy verbatim the equations:

```
1. a+b+c=1; a+2b+3c=1; 3a+5b+9c=1;
2. showvariable a,b,c;
```

Running METAFONT on this program results in the following message:

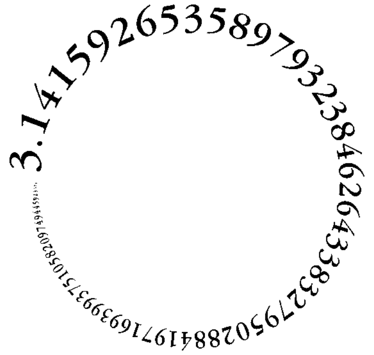
```
a=0
b=2
c=-1
```

The message is due to the statement `showvariable`. The statement `message` might have been used as well, but then numbers should be converted to strings using the operation `decimal` (see the previous two pages).

Solving such problems using METAFONT does not seem too practical, unless help in doing a child's homework is needed... Nonetheless, METAFONT's talents are not to be ignored. It is worth mentioning that thousands of equations do not frighten METAFONT.[4] Matrices of this form arise as a result of discretization of partial differential equations. The right-hand sides of the equations were chosen in such a way that the exact solution was given by $x_i = \frac{1}{10}i$. The average square error was about 0.025 for $n = 1000$, about 0.65 for $n = 2000$, and about 4.95 for $n = 4000$; maximal errors were about 0.036, 1.00, and 6.87, respectively. The calculations lasted 40″, 3′ 10″, and 13′ 45″, respectively (an IBM PC compatible, 486 processor). It shows the strength and the weakness of METAFONT's numerical machinery.

My intention was to show a genuinely impractical application. Eventually, the reader is to decide whether I hit the target. Note, however, that a neat example of a METAFONT calculator can be found in *The* METAFONT*book* (the program `expr.mf`, p. 61). D. E. Knuth admits, that he occasionally uses METAFONT as a pocket calculator — why not follow the master? After all, calculators can solve also systems of linear equations...

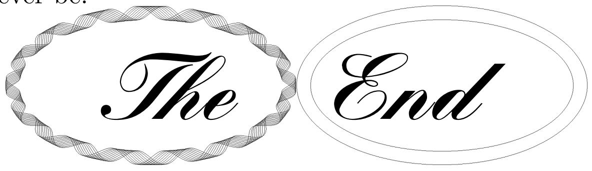
Recreational applications. Finally, let's have a look at two examples of figures that can be produced using METAFONT. This time, only the results will be presented, otherwise the reader might be bored stiff.



[4] A few details for math-oriented users: systems of linear equations defined by matrices $[a_{i,j}]$, $i = 1, 2, \ldots, n$, $j = 1, 2, \ldots, n$, such that $a_{i,i} = 4$, $a_{i,j} = -1$ for $i - j = 1$ or $i - j = 25$, $a_{i,j} = 0$, were tested for $n = 1000, 2000, 4000$.

I borrowed the idea of winding the number π around a circle from Alan Hoenig. The fractal "branches" were published in "PostScript Language Journal", **2**, No. 4. Translation from PostScript to METAFONT and back is an instructive and thus an advisable exercise, indeed.

One might call such applications "applications of amusement". I would reply that amusement is no sin. On the contrary, it is often truly inspiring, perhaps even more than serious applications can ever be.



◇ Bogusław Jackowski
BOP s.c., ul. Piastowska 70,
Gdańsk, Poland
`B.Jackowski@gust.org.pl`

Language Support

Typesetting Bengali in TEX

Anshuman Pandey

1 Introduction

The Bengali (or *Bāṅglā* বাংলা) script is one of the thirteen primary scripts used throughout India. Like other Indic scripts, the Bengali is derived from the ancient Brahmi script. The script is intimately tied with the Bengali language, which according to the latest data from *Ethnologue*, is currently the fourth most spoken language in the world with roughly 189 million speakers [2]. The language is spoken mainly in the Indian province of West Bengal and in Bangladesh. Bengali has been the medium for many notable artists, of whom the famous *littera-teur* Rabindranath Tagore and the great film-maker Satyajit Ray are the best known.

2 The Script

Of all the scripts derived from Brahmi, Bengali is most closely related historically to Devanagari. The two scripts share a comparable inventory of consonant, vowel, and conjunct characters, however, aside from superficial form and design, the primary difference is the phonetic value assigned to certain characters of the Bengali script.[1]

Like all Brahmi-based scripts, Bengali is technically an alpha-syllabic script. This system is based on the unit of the "graphic" syllable, or *akṣara*, which by definition always ends in a vowel. Each basic consonant character in Bengali is understood to represent the consonant modified by the inherent vowel *a*, *eg.* ক = *ka*.

When a consonant is modified by any other vowel, the syllable is written using a diacritic form of the vowel. For example, the syllable *kā* কা is composed of the consonant *ka* ক and the vowel *ā* আ (diacritic form: া). Syllables consisting of only a vowel, or with a vowel in word-initial position, are written with the full form of the vowel.

A "graphic" syllable consisting of a sequence of consonants is written using a specific conjunct form, or *ẏuktākṣar* যুক্তাক্ষর . For instance, the consonant cluster *kka* (ক্ + ক) is written as a single grapheme, ক্ক, not laterally as কক. The latter form

[1] For further details on Bengali phonology please consult Suniti Kumar Chatterji's *Origin and Development of the Bengali Language*, Allen & Unwin: London, 1970–72, reprint of 1926 ed.

represents *kaka*, not *kka*. Such conjuncts are relatively easy to decipher because the conjunct retains some semblance to the individual elements it comprises. However, with others decipherment is a little more daunting because the form gives no indication of its constituents: *kṣa* (ক + ষ) ক্ষ, *ṅga* (ঙ + গ) ঙ্গ, and *tra* (ত + র) ত্র.

The subscript character called *hasanta* (্) is used to indicate an elision of the inherent *a* from a consonant. For example, ক্ is *k*, not *ka*. In some instances, generally when dealing with poor or limited types, the *hasanta* is used in modifying consonants to produce conjuncts. The modified consonant is then written laterally with the following consonant to produce a simplified ligature. Therefore, ক্ক may theoretically be written as ক্‌ক, but such practice is rare as traditional Bengali orthography places great importance on the proper formation of letters.

3 Short History of Bengali Typesetting

The development of modern modes of printing and typesetting Bengali coincided with the assumption of government of Bengal by the British East India Company in 1772. Seeing the need to educate its officials in the vernacular, the British sought to provide a means by which to expedite such instruction. The task was taken up by Nathaniel Brassey Halhed who wrote a book titled *A Grammar of the Bengal Language*. Printed in 1778 at Hoogly (near Calcutta), this was the first book containing Bengali characters printed with movable types. These movable types were cast by Sir Charles Wilkins.

Wilkins, considered the pioneer of Indic typesetting, not only instituted the mechanical printing of Bengali, but even trained native technicians and motivated them to apply their skills to other Indic scripts as well. One of Wilkins' students, a man by the name of Panchanana Karmakar, eventually brought forth a large inventory of Bengali types which led to further advancements in the printing of Bengali [5].

4 Bengali Fonts and Packages for TEX

The typesetting of Bengali gained another big boost when it was introduced to electronic typesetting and publishing. Initially, due to the complex nature of Bengali conjuncts and the intricate design elements of the basic characters themselves, it was rather difficult to find a computer font containing a complete set of conjuncts for the script, and/or a typeface which was aesthetic in its display of the intricate and complex glyphs.

TEX was first introduced to Bengali in 1992 when Avinash Chopde added a support module for the HP Softfont 'SonarGaon' (sgaon) to his itrans package. 'SonarGaon' was designed by Anisur Rahman in 1990, and was available in three weights: normal, slanted, and compressed. Although it was a decent Bengali font for the time, it was less than complete in that it lacked several standard ligatures.

To amend this deficiency, Muhammad Ali Masroor developed a package which supplemented the support for 'SonarGoan' in itrans. Called arosgn, or "Aro SonarGaon" ("more SonarGaon"), this package extended the 'SonarGaon' font by providing in a METAFONT source the glyphs missing from the parent font and a set of macros with which to access these new extensions.

The support for 'SonarGaon' and arosgn was later dropped from itrans partially because the font was retracted from the public domain. The other factor was the development by Shrikrishna Patil of a user-defined Type-1 PostScript font called "ItxBengali". This font was adapted for use with itrans and replaced 'SonarGoan'.[2]

Last, but not least, is a program called 'Bengali Writer' developed by Abhijit Das (Barda).[3] 'Bengali Writer' is a text editor for the X11 windows system which allows the user to type Bengali documents and save them in TEX format. Documents saved in TEX format are then to be used with TEX in conjunction with the 'Bengali Writer TEX Interface' (bwti) package.

The bwti package consists of a beautiful Bengali METAFONT and a set of macros which facilitate the inputting of Bengali text. The input system is not extremely fluid and has low readability. It appears to be a verbatim reflection of the font encoding, and conjuncts are defined as macros. For example, with bwti a word like *digbijaẏa* দিগ্বিজয় must be input as `idi\gb jy` instead of `digbija.ya` as with the bengali package.

5 The bengali Package

The *Bengali for TEX* (bengali) package is housed on CTAN in the `language/bengali/pandey/` directory. This package differs from those described in the above section in that it:

1. provides a means aside from itrans and bwti of typesetting Bengali
2. provides a simple, stand-alone preprocessor interface

[2] These and other Indic language packages are described in the article "An Overview of Indic Fonts for TEX" [4].

[3] For more information on 'Bengali Writer' and related utilities see `http://www2.csa.iisc.ernet.in/~abhij/bwti/`.

3. implements the 'Velthuis' transliteration scheme (see Section 5.2 for details)
4. provides a simple, macro-based method of delimiting Bengali text, similar in form and style to other Indic script packages like *Devanagari for TEX*
5. complies with the New Font Selection Scheme (NFSS)
6. incorporates the latest version of the 'Bengali' METAFONT developed by Abhijit Das (Barda)

5.1 The 'Bengali' Font

Das designed his 'Bengali' METAFONT after studying the various types employed in the modern printing of Bengali books in India. Das emphasized that he did not model the font after the design and form of any particular typeface, nor did he employ any instruments in its development, but relied heavily on crude approximations made by the naked eye. Upon close inspection, the characters of Das's METAFONT adhere to the traditional Bengali orthographic style found in many printed books, and rival many modern Bengali typefaces of the highest quality.

The beautiful 'Bengali' (bn) font is currently available in two shapes: normal (bnr) and slanted (bnsl). It contains an almost complete inventory of Bengali vowels, consonants, numerals, diacritics, and punctuation marks. The font does lack three rare characters: r̥̄ (long vocalic 'r'), l̥ (vocalic 'l', resembling the numeral 9: ৯), and l̥̄ (long vocalic 'l').[4]

Nonetheless, the font contains the entire repertoire of traditional Bengali conjunct and ligature forms, and also a few extra ligatures used in writing loan words. In addition to the conjunct forms given in Table 2, Bengali has a few special consonant-vowel ligatures formed from the vowel *u* উ, and in one case the vowel r̥ ঋ. These are illustrated in the chart below:

গ + উ	`gu`	গু	হ + উ	`hu`	হু
র + উ	`ru`	রু	ত + র + উ	`tru`	ত্রু
র + উ	`ruu`	রূ	ন + ত + উ	`ntu`	ন্তু
শ + উ	`"su`	শু	ল + গ + উ	`lgu`	ল্গু
হ + ঋ	`h.r`	হৃ	স + ত + উ	`stu`	স্তু

[4] The existence of these characters in the traditional Bengali syllabary is debatable. Although *The Unicode Standard, Version 2.0* does not place the long forms of these characters in the traditional ordering, it does reserve the positions they would occupy were they to be included in the proper arrangement. Nor does it list them under the heading 'Independent Vowels', but places them instead under 'Generic Additions.' Unicode also gives the diacritic form of these characters [7]. However, the ISO draft standard 15919 for the transliteration of Indic scripts does show the two long forms in the traditional arrangement [6].

With a few exceptions — namely *kra* ক্র, *tra* ত্র, and *bhra* ভ্র, and variants of these — Table 2 does not show conjuncts formed with *ra* র or *ya* য, as most ligatures containing these two elements are produced in a relatively static vertical or lateral fashion. In the case of *ra*, the *ra-phala* (্র), or the ligature form of `ra` is placed beneath the consonant: প্র `pra` and স্র `sra`. In the case of `ya`, the *ya-phala* (্য) is placed after the consonant: প্য `pya` and স্য `sya`.

Additionally, when *ra* র is the initial element in a consonant conjunct, it is written as a superscript diacritic called *repha*. For instance, the word *karma* is written কর্ম `karma`, with the *r* above the *ma*.

As there are no character primitives in the bn font, producing conjuncts containing non-traditional phonemes is a problem. A few of these have been accounted for, but there may be cases which have been overlooked. One solution is to use the *hasanta* to explicitly produce such characters. If the preprocessor detects a conjunct it does not recognize it will 'create' one by joining the full sizes with a *hasanta*. The result is not pretty, but it is a solution!

A few subtle kerning adjustments were needed to correctly align the placement of vowel diacritics below consonants. The preprocessor manages such kernings and therefore manual adjustments are not needed.

Since the font contains several complex ligatures which will produce rather unsatisfactory output at small sizes such as 8pt, font magnifications below 10pt are not supported. Therefore attempts to use bn in footnotes and with such NFSS commands as `\small` will result in a message from TEX complaining about a missing metric file. Perhaps a solution is possible in the near future.

5.2 Transliterated Input

Bengali text is entered in transliteration. Each character in the Bengali syllabary has been assigned a corresponding value based on the Roman alphabet, or a combination of signs from the Roman alphabet. The author has adapted the 'Velthuis' transliteration scheme for the bengali package.

The 'Velthuis' scheme was developed in 1990 by Frans Velthuis as a means of providing transliterated input for his *Devanagari for TEX* package, or devnag. As the Bengali syllabary resembles the Devanagari, the 'Velthuis' scheme is perfectly suitable to transliterate Bengali (and, for that matter, any other Indic script). It was necessary to add a few extensions to the scheme, otherwise it has remained largely unchanged from the original. The scheme as modified for Bengali is given in Table 1.

STOPS

	C	V
ক	*ka*	`ka`
খ	*kha*	`kha`
গ	*ga*	`ga`
ঘ	*gha*	`gha`
ঙ	*ṅa*	`"na`
চ	*ca*	`ca`
ছ	*cha*	`cha`
জ	*ja*	`ja`
ঝ	*jha*	`jha`
ঞ	*ña*	`~na`
ট	*ṭa*	`.ta`
ঠ	*ṭha*	`.tha`
ড	*ḍa*	`.da`
ড়	*ṛa*	`Ra`
ঢ	*ḍha*	`.dha`
ঢ়	*ṛha*	`Rha`
ণ	*ṇa*	`.na`
ত	*ta*	`ta`
থ	*tha*	`tha`
দ	*da*	`da`
ধ	*dha*	`dha`
ন	*na*	`na`
প	*pa*	`pa`
ফ	*pha*	`pha`
ব	*ba*	`ba`
ভ	*bha*	`bha`
ম	*ma*	`ma`

SONORANTS

	C	V
য	*ya*	`ya`
য়	*ẏa*	`.ya`
র	*ra*	`ra`
ৰ	*ra*	`~ra`
ল	*la*	`la`
ব	*ba*	`ba`
ৱ	*va*	`va`
শ	*śa*	`"sa`
ষ	*ṣa*	`.sa`
স	*sa*	`sa`
হ	*ha*	`ha`

VOWELS

		C	V
অ		*a*	`a`
আ	◌া	*ā*	`aa`
ই	◌ি	*i*	`i`
ঈ	◌ী	*ī*	`ii`
উ	◌ু	*u*	`u`
ঊ	◌ূ	*ū*	`uu`
ঋ	◌ৃ	*r̥*	`.r`
এ	◌ে	*e*	`e`
ঐ	◌ৈ	*ai*	`ai`
ও	◌ো	*o*	`o`
ঔ	◌ৌ	*au*	`au`

NUMERALS

	C	V
০	*0*	`0`
১	*1*	`1`
২	*2*	`2`
৩	*3*	`3`
৪	*4*	`4`
৫	*5*	`5`
৬	*6*	`6`
৭	*7*	`7`
৮	*8*	`8`
৯	*9*	`9`

SPECIAL CHARACTERS

	C	V	
ং	*ṃ*	`.m`	*anusvāra*
ঁ	*ṁ*	`~m`	*candrabindu*
ঃ	*ḥ*	`.h`	*visarga*
্		`&`	*hasanta*
ৎ	*t*	`T`	*khaṇḍa ta*
।	.	`|`	*dāṁṛi*

C	CSX+ 8-bit scheme
V	‘Velthuis’ 7-bit scheme

Table 1: Inventory of Bengali Characters

ক + ক	kk	ক্ক	ত + থ	tth	ত্থ	ম + ভ + র	mbhr	ম্ভ্র
ক + ট	k.t	ক্ট	ত + ন	tn	ত্ন	ম + ম	mm	ম্ম
ক + ত	kt	ক্ত	ত + ব	tb	ত্ব	ম + ল	ml	ম্ল
ক + ম	km	ক্ম	ত + ম	tm	ত্ম	ল + ক	lk	ল্ক
ক + র	kr	ক্র	ত + র	tr	ত্র	ল + গ	lg	ল্গ
ক + ল	kl	ক্ল	থ + ব	thb	থ্ব	ল + ট	l.t	ল্ট
ক + ব	kb	ক্ব	দ + গ	dg	দ্গ	ল + ড	l.d	ল্ড
ক + ষ	k.s	ক্ষ	দ + ঘ	dgh	দ্ঘ	ল + প	lp	ল্প
ক + ষ + ণ	k.s.n	ক্ষ্ণ	দ + দ	dd	দ্দ	ল + ব	lb	ল্ব
ক + ষ + ম	k.sm	ক্ষ্ম	দ + ধ	ddh	দ্ধ	ল + ম	lm	ল্ম
ক + স	ks	ক্স	দ + ব	db	দ্ব	ল + ল	ll	ল্ল
গ + ধ	gdh	গ্ধ	দ + ভ	dbh	দ্ভ	শ + চ	"sc	শ্চ
গ + ন	gn	গ্ন	দ + ভ + র	dbhr	দ্ভ্র	শ + ছ	"sch	শ্ছ
গ + ম	gm	গ্ম	দ + ম	dm	দ্ম	শ + ন	"sn	শ্ন
গ + ল	lm	গ্ল	ধ + ন	dhn	ধ্ন	শ + ম	"sm	শ্ম
গ + ব	gb	গ্ব	ধ + ব	dhb	ধ্ব	শ + ল	"sl	শ্ল
ঘ + ন	ghn	ঘ্ন	ন + ত	nt	ন্ত	শ + ব	"sb	শ্ব
ঙ + ক	"nk	ঙ্ক	ন + ত + ব	ntb	ন্ত্ব	ষ + ক	.sk	ষ্ক
ঙ + ক + ষ	"nk.s	ঙ্ক্ষ	ন + ত + র	ntr	ন্ত্র	ষ + ক + র	.skr	ষ্ক্র
ঙ + খ	"nkh	ঙ্খ	ন + থ	nth	ন্থ	ষ + ট	.s.t	ষ্ট
ঙ + গ	"ng	ঙ্গ	ন + দ	nd	ন্দ	ষ + ঠ	.s.th	ষ্ঠ
ঙ + ঘ	"ngh	ঙ্ঘ	ন + দ + ব	ndb	ন্দ্ব	ষ + ণ	.s.n	ষ্ণ
ঙ + ম	"ngm	ঙ্ম	ন + ধ	ndh	ন্ধ	ষ + প	.sp	ষ্প
চ + চ	cc	চ্চ	ন + ন	nn	ন্ন	ষ + ফ	.sph	ষ্ফ
চ + ছ	cch	চ্ছ	ন + ব	nb	ন্ব	ষ + ম	.sm	ষ্ম
চ + ছ + ব	cchb	চ্ছ্ব	ন + ম	nm	ন্ম	স + ক	sk	স্ক
চ + ঞ	c~n	চ্ঞ	ন + স	ns	ন্স	স + ক + র	skr	স্ক্র
জ + জ	jj	জ্জ	প + ট	p.t	প্ট	স + ক + ল	skl	স্ক্ল
জ + জ + ব	jjb	জ্জ্ব	প + ত	pt	প্ত	স + খ	skh	স্খ
জ + ঝ	jjh	জ্ঝ	প + ন	pn	প্ন	স + ট	s.t	স্ট
জ + ঞ	j~n	জ্ঞ	প + প	pp	প্প	স + ত	st	স্ত
জ + ব	jb	জ্ব	প + ল	pl	প্ল	স + ত + র	str	স্ত্র
ঞ + চ	~nc	ঞ্চ	প + স	ps	প্স	স + থ	sth	স্থ
ঞ + ছ	~nch	ঞ্ছ	ফ + ল	phl	ফ্ল	স + ন	sn	স্ন
ঞ + জ	~nj	ঞ্জ	ব + জ	bj	ব্জ	স + প	sp	স্প
ঞ + ঝ	~njh	ঞ্ঝ	ব + দ	bd	ব্দ	স + প + ল	spl	স্প্ল
ট + ট	.t.t	ট্ট	ব + ধ	bdh	ব্ধ	স + ফ	sph	স্ফ
ট + ব	.tb	ট্ব	ব + ব	bb	ব্ব	স + ব	sb	স্ব
ড + ড	.d.d	ড্ড	ব + ল	bl	ব্ল	স + ম	sm	স্ম
ণ + ট	.n.t	ণ্ট	ভ + র	bhr	ভ্র	স + ল	sl	স্ল
ণ + ঠ	.n.th	ণ্ঠ	ভ + ল	bhl	ভ্ল	হ + ণ	h.n	হ্ণ
ণ + ড	.n.d	ণ্ড	ম + ন	mn	ম্ন	হ + ন	hn	হ্ন
ণ + ণ	.n.n	ণ্ণ	ম + প	mp	ম্প	হ + ম	hm	হ্ম
ণ + ম	.nm	ণ্ম	ম + ফ	mph	ম্ফ	হ + ব	hb	হ্ব
ত + ত	tt	ত্ত	ম + ব	mb	ম্ব	হ + ল	hl	হ্ল
ত + ত + ব	ttb	ত্ত্ব	ম + ভ	mbh	ম্ভ	ড় + গ	Rg	ড়্গ

Table 2: Supported Bengali conjuncts

কে লইবে মোর কার্য, কহে সন্ধ্যা রবি
শুনিয়া জগৎ রহে নিরুত্তর ছবি ।
মাটির প্রদীপ ছিল, সে কহিল, স্বামি
আমার যেটুকু সাধ্য করিব তা আমি ।
— রবিন্দ্রনাথ ঠাকুর

```
{\bn ke la{}ibe mor kaarya, kahe sandhyaa rabi
"suni.yaa jagaT rahe niruttar chabi |
maa.tir pradiip chila, se kahila, sbaami
aamaar ye.tuku saadhya kariba taa aami |
                         -- rabindranaath .thaakur}
```

Figure 1: A Poem by Rabindranath Tagore

Bengali does not distinguish between *ba* and *va*, and collapses both into *ba*. For this reason, as a general rule, all words containing *va* should be transliterated as *ba*. However, when parsing conjuncts the preprocessor will accept *va* in place of *ba*, so the word স্বামি may be transliterated as either `svaami` or `sbaami`. If the preprocessor detects *va* outside of a conjunct, it will assume that the Assamese *va* is intended, and will produce an undesired output.

Another significant point is the use of `{}` to break the lexical scan. This is important when two short vowels are encoded in succession. The brackets will prevent the preprocessor from interpreting the two short vowels as a diphthong. For example, compare লইবে `la{}ibe` with লৈবে `laibe`.

The above two points are illustrated in Figure 1. As indicated in the figure, the transliterated input text is placed within the scope delimited by `{\bn` and `}`. This allows the preprocessor to locate and properly translate the input to the appropriate font character codes.

5.3 The Macros

The initialization for the bengali package is managed by the beng style file. The style defines:

1. the macro `\bn` as the delimiter for Bengali text. The definition of `\bn` sets the current font to the bn family, initializes bn for use with NFSS, and provides an appropriate `\baselineskip` so there is adequate clearance between lines for super- and subscript characters.
2. a bengali counter which may be used to produce Bengali numerals for page numbering and enumerated environments.

An idiosyncracy of the `\bn` macro is that a curly brace must immediately precede it and that a space must immediately follow it: `{\bn␣`. Otherwise, the preprocessor will return an error stating it was unable to locate a valid Bengali delimiter.

The document containing Bengali text is to be considered as any other TeX file. As Bengali text is delimited by `{\bn` *text* `}`, any other TeX macros and packages may be used in the document. However, these Bengali documents should have the extension `.bn` in the file name. This lets the preprocessor know that the file is associated with it.

Additionally, any shape- or size-changing commands such as `\large` or `\slshape` should follow the `\bn` macro. Otherwise, the declaration of the default font as bn will be over-written by the NFSS macro's default declaration of the Computer Modern fonts. The result will be a jumble of characters, not the intended Bengali.

5.4 The Preprocessor

Once the Bengali text has been transliterated, the file is run through a program called a preprocessor. A preprocessor is the ideal method for enabling the typesetting of such complex scripts as Bengali because it presents the user with a simple interface for character input. The user has only to be concerned with the transliteration, because the preprocessor will manage the conversion of the input text into character codes with which TeX is familiar.

A preprocessor called beng has been developed for use with the bengali package. It is a small program written in C and based in function on the preprocessor for Charles Wikner's sanskrit package. The syntax for its use is:

`beng` *infile*[`.bn`] [*outfile*[`.tex`]]

where *infile* is the name of the Bengali document. The `.bn` extension is optional, as is the target output file. By default the preprocessor will name the output file the same as the input, but with the extension `.tex`.

Running the preprocessor without any arguments invokes the interactive mode. The input and output filenames must then be manually entered at the prompts. The version number and other information can be obtained from the preprocessor by invoking it with `-h`. The author's email address is also given in case any problems arise with the use of the program.

5.5 A Note on Hyphenation

There is no tradition of hyphenating words in Indian orthography. In manuscripts, the use of hyphens is non-existent, and in early printed materials, hyphenation was applied arbitrarily between any *akṣara*-s when the end of a line was reached. Nowadays, printers are more keenly aware of word-breaks at the end of lines, and attempt to maintain syntactic sense when applying hyphenation.

Unfortunately, such "intelligent" hyphenation cannot be produced with the bengali package. When the preprocessor converts transliterated input into internal character codes, the output deviates substantially from what TeX would consider a 'word'. However, if by chance a word is broken at the end of a line, TeX will produce a hyphen because the bn font possesses a hyphen character at the standard position. A hyphen may otherwise be encoded within a given text simply by typing the hyphen character: `-`.

6 Support for Assamese

The script used in the far eastern Indian province of Assam is nearly identical to the Bengali script. The Assamese (or *Asamīya* অসমীয়) script (also the name of the associated language) differs from the Bengali in the design of two consonant characters, although the correspondence between pronunciation and script is also different in a number of respects between Assamese and Bengali. These two characters are *ra* ৰ and *va* ৱ.

These characters were not part of the original bn font developed by Das. The author has created them based on the design found in Halhed's Bengali grammar [3] and the descriptions given by Banerji [1].

It is interesting that these two characters are used in Halhed's grammar. Their appearance in the book implies that they were commonly used in Bengali orthography, and that the switch from ৰ to র and from ৱ to ব must have occurred within the past 200 years. Also, it is probably during this time that these earlier forms were restricted to the Assamese script.

A group led by Jugal Kalita is believed to be working on an Assamese package.[5] However, no indication of progress has been posted at the project's site, and as of yet, nothing has been released.

[5] Details are available at `http://www.acsu.buffalo.edu/~talukdar/assam/language/assamlang.html`.

7 What's Next?

7.1 Refining the font

In addition to creating a suitable boldface, Das and the author have discussed replacing the explicit conjunct glyphs with character primitives. One advantage of using primitives is the opening up of several positions for other characters. The disadvantage is replicating traditional ligature forms with primitives: ligatures can be formed laterally or by juxtaposition, but this method serves an injustice to the aesthetic of the Bengali script.

Depending on the number of character positions available after the font has been revised, new characters may be introduced to the repertoire. Possible additions may be the characters *r̥̄*, *l̥*, and *l̥̄*, along with their diacritic forms (described in further detail in Section 5.1). Other possible inclusions may be the currency signs given in Unicode.

Taco Hoekwater has generously offered to produce Type-1 versions of the bn font. Once the font is stabilized, the conversions will be performed and made available as part of the bengali package.

7.2 Uniting babel and bengali

Jun Takashima has developed hyphenation patterns for Sanskrit and Kannada to be used in conjunction with Johannes Braams's babel package. As the lexica of both Kannada and Bengali are heavily influenced by Sanskrit, it is feasible that either the Sanskrit or Kannada hyphenation pattern may be adapted for use with the Bengali language.

Such an adaptation would require an input encoding scheme compatible with the babel convention. One idea could be to modify the beng preprocessor to convert the 'Velthuis' transliterated input to this babel-compliant scheme.

7.3 Printing Dates in Bengali

A new method for printing dates in Bengali needs to be developed. This new method would print the date according to both the Western and traditional Bengali calendars. Both styles are commonly used throughout Bengali-speaking regions of the world.

Of the two, the Western style is easier to implement. As is expected, the `\today` macro does not produce the correct result in the Bengali environment. To overcome this, Masroor, who also developed the arosgn package, wrote a LaTeX package called bngtoday, intended for use with itrans.[6] This package provides the macro `\BanglaToday` which gives the current Western date transliterated into Bengali. For example, June is simply *Jūn* জুন.

[6] On CTAN at `language/bengali/bngtoday.sty`

The traditional Bengali calendar is quite different from the Western and therefore requires a bit more work for correct implementation. Due to the arrangement of the Bengali calendar, Bengali and Western months overlap. Thus, a given Western month may be known by two different names in Bengali. For example, June may either be *Jyaiṣṭha* জ্যৈষ্ঠ or *Āsāṛ* আসাড়, depending on whether the first or second half of the month is being referred to. Furthermore, জ্যৈষ্ঠ may also refer to the last half of May, and আসাড় to the first half of July.

Additionally, in the Bengali calendar the first four days of the month have special names, and ordinal numbers used for days of the month also have distinct forms. The new package would automate the calculations needed to produce the date according to these conventions.

7.4 Implementing CSX+ in bengali

CSX+, or Classical Sanskrit eXtended+, is an 8-bit encoding scheme which parallels the convention adopted for the ISO Committee Draft 15919 [6]. Contrary to what its name indicates, CSX+ uniformly supports all Indic scripts. The CSX+ scheme is modelled after IBM Code Page 437 and occupies characters in the Upper ASCII block. Table 1 shows the CSX+ scheme as it pertains to Bengali.

Currently the `beng` preprocessor only recognizes 7-bit input in the form of the 'Velthuis' transliteration scheme. The author intends to extend the preprocessor to recognize the 8-bit CSX+ encoding scheme as well.

8 Notes

The author was informed of a book by Fiona G. E. Ross titled *The Printed Bengali Character* (Curzon: Richmond, 1999) which provides lucid and detailed information on Bengali orthography and typography. The author was strongly encouraged to review Ross's work, but unfortunately, a copy of the book could not be acquired in time. It is hoped that this does not diminish the force of the article.

References

[1] Banerji, R. D. *The Origin of the Bengali Script.* Nababharat Publishers: Calcutta, 1973. Reprint of 1973 1st ed.

[2] Grimes, Barbara F. [ed]. *Ethnologue: Languages of the World.* Summer Institute of Linguistics, Inc. Dallas, Texas, 1996. [**Note:** Information on most widely spoken languages may be found at `http://www.sil.org/ethnologue/`. The list was last updated in February 1999.]

[3] Halhead, Nathaniel Brassey. *A Grammar of the Bengal Language.* Printed at Hoogly, Bengal, 1778.

[4] Pandey, Anshuman. "An Overview of Indic Fonts for TeX", *TUGboat*, **19**(2), 1998. pp. 115–120.

[5] Priolkar, Anant K. *The Printing Press in India: Its Beginnings and Early Development* [Being a quatercentenary commemoration study of the advent of printing in India (in 1556)]. Marathi Samshodhana Mandala: Bombay, 1958.

[6] Stone, Anthony [ed]. *ISO Committee Draft 15919: Transliteration of Devanagari and Related Scripts into Latin Characters.* Available at `http://ourworld.compuserve.com/homepages/stone_catend/trdcd1c.htm`.

[7] The Unicode Consortium. *The Unicode Standard, Version 2.0.* Addison-Wesley Developers Press: Reading, Massachusetts, 1997.

◇ Anshuman Pandey
University of Washington
Department of Asian Languages and Literature
225 Gowen Hall, Box 353521
Seattle, WA 98195
`apandey@u.washington.edu`
`http://weber.u.washington.edu/~apandey/`

Software & Tools

The *CTAN May 1999* CD ROM set by DANTE e.V. and Lehmanns bookstore

Klaus Höppner

1 About the CD ROM set

This year, DANTE e.V. produced the snapshot of CTAN in cooperation with Lehmanns Fachbuchhandlung, a German bookstore — under participation of TUG and several international user groups who are giving the CD ROM set to their members.

The set contains a nearly complete snapshot of CTAN on 3 CD ROMs that was taken from `dante.ctan.org` on May 30th, 1999. Since the archive size of CTAN was about 3 GB, some parts had to be zipped or left out, respectively.

Zipped Directory trees	**Missing Directories**
fonts	obsolete
digests	nonfree/support/adobe
web	support/ghostscript
parts of systems	

The CD ROM set includes some additions from early June, like the new teTEX version 1.0(.5) and fpTEX 0.3e. Additionally, the current ConTEXt and PPCHTEX macros from `http://www.pragma-ade.nl` are included.

2 Technical remarks

The CD ROM snapshot from CTAN was made using the programs mkisofs and cdrecord under Linux. The CD ROMs use the ISO 9660 file system with Rock Ridge and Joliet extensions. Thus, the majority of computers running under UNIX or Microsoft Windows 9x/NT should be able to display long filenames.

Computers running under operating systems not supporting Rock Ridge or Joliet extensions for CD ROMs will only display 8.3 file names. Each directory contains a file `TRANS.TBL` that relates the short file names to the long ones.

All text files on the CD ROMs use the UNIX style line endings (LF). This can cause problems with some editors under operating systems using different line end conventions. One possible solution is to use the Info-ZIP programs by compressing a file with “zip” and uncompressing it with “*unzip -a*” where the proper line endings for the used operating system are automatically produced.

3 Important note for users of Windows 9x/NT

Due to an error in the version of MKISOFS that was used to create the CD images, Windows Explorer will display a file `<translation table>` in each directory of the CD ROM that can't be opened. Unfortunately, this prevents directory trees from being copied from the CD ROM to the hard disk drive by drag and drop in Windows Explorer. Windows Explorer will stop copying of all files with the error message that it can't copy the file `<translation table>`.

To copy directory trees from CD ROM to hard disk, please proceed as follows:

1. Open a command prompt (Start → Programs → Command Prompt)
2. Copy the directory tree with
   ```
   xcopy /s /c source-directory
   target-directory
   ```
   (of course in *one* line)

   Example: If your CD ROM drive is `D:` and you want to copy the directory `macros\latex\required` from CD ROM to the directory `foo` on your hard disk drive (`C:`), please type
   ```
   xcopy /s /c d:\macros\latex\required
   c:\foo
   ```
   (as above in *one* line)
3. If the target directory doesn't exist, you will be prompted whether the target is a file or a directory. Please type `D` for directory in this case. (This letter is only valid for the English version of Windows!)
4. During copying you will get warnings that the file `<translation table>` can't be copied. You can ignore these warnings.
5. You will get additonal information about the usage of `xcopy` by typing
   ```
   xcopy /?  | more
   ```

4 The cover of the CD ROM set

Usually, the CD ROM set is shipped in a digifile cover. TUG decided to order the sets without it. For this reason I created a PDF file (see page 84) based on the original cover that can be downloaded from `http://www.tug.org/texlive/cover.pdf`, and should fit into a standard jewel case. Of course, it looks nicer if you have access to a color printer...

◇ Klaus Höppner
University of Dortmund
Institute of Accelerator Physics and Synchrotron Radiation
D-44221 Dortmund, Germany
`k.hoeppner@physik.uni-dortmund.de`

Interacting pdfTEX, PERL and ConTEXt

Gilbert van den Dobbelsteen

Abstract

PERL, pdfTEX, and ConTEXt are extremely useful in the production of large documents which also need a lot of interaction. This article resulted from a job I did for a good friend, yielding over 2000 pages of PDF output.

The power here is to use the right tool for the right job. Almost everything created for this job could be done in TEX, but since I am just a 'Ben Lee User', I use different tools to get the job done. So it is not a matter of which tool is the best for the job, but more like *Which tool is best for the person using the tool*.

1 Introduction

A few months ago, a good friend (let's call him Bart, because that's his name) had a problem. He had taken on a job where he needed to create an interactive document consisting of over a thousand paragraphs. All texts needed to be clickable, and as a result a poster should pop-up with the same text, but artistically enhanced. The texts originated from the LOESJE association and so did the posters.

I advised him to take a look at ADOBE ACROBAT. He did, and he had already made a framework with some buttons and clickable links. He started calculating, and decided this was too much work. Every link had to be manually created, and since each poster-text had about three to four categories, this meant drawing over 5000 clickable areas by hand in ACROBAT. Though LOESJE has many volunteers, you can't give them such a boring job. It would simply kill the relaxed atmosphere, normally hanging around LOESJE.

So I told him that he could probably use some programming tool to automate things. Since Bart is dyslectic (it is very difficult for him to read words from paper or screen), he is unable to do classical software engineering jobs, so in the end I volunteered to do the job for him.

I usually write small documents, which aren't larger then 100 pages, but I was very sure TEX is capable of doing larger ones. Interactive programs usually have big problems dealing with large files and many pages, but since TEX is batch oriented I knew this wasn't going to be a problem.

This article, based on a paper presented at the meeting of 1–3 May 1999 in Bachotek, Poland, appeared in the GUST Biuletyn, **12** (1999), pp. 64–69, and appears here by permission.

2 About LOESJE

LOESJE is an association of people who design strong texts for different applications. Some text-categories are: Elections, Politics, Year 2000 problems, Astrology, Economy, Stock exchange, Christmas, Nature, Animals, Poetry, Religion, School, Health-care, et cetera.

These texts are put onto posters and flyers and you can see them anywhere around the Netherlands. You can also buy post-cards and other stuff.

The main idea is to trigger people to think about what is going on. A typical text from LOESJE:

Year 2000: Suppose the end of the world is near and God forgot to make a backup

The LOESJE association has been around since 1983, and throughout the years they have created 1350 different texts.

To celebrate their 15-year existence, they decided to create a CD-ROM with all their posters on it, and with a nice catalogue, where you can browse the texts category-wise or chronologically.

3 How things got started

I had to define some structure before I could begin. In the beginning of LOESJE they used markers and pencils to create posters by hand, and reproduced them with a large xerox machine. So those posters weren't available in a digital format. They started using computers many years later, so much of the material was only available on paper.

To assure quality and consistent presentation they decided to scan all posters. LOESJE has a Scanjet 2 and lots of volunteers. The scanner was old and the compressed TIFF output generated TIFF files with errors, so they had to fall back to uncompressed TIFF.

After a few weeks Bart came to me with 10 CDs full of uncompressed TIFF bitmaps. Each file was 4Mb in size consisting of a 600DPI A4 scan of each poster. This started to terrify me. My computer had about 3Gb of free disk space, which was definitely not enough for ten CDs of data. How to proceed? I knew that I needed the files in PNG format for inclusion in pdfTEX. So I decided to convert all files to PNG with the ImageMagick tools. This took 8 hours of computer time and in the end I discovered the dimensions where lost in the resulting PNG-file. After investigating the originals I concluded the dimensions weren't present there either.

Since the PNG format is compressed, and the monochrome scans are very large, the total size reduced from 10 CDs to $\frac{1}{6}$th CD. This was a manageable amount of data.

3.1 Texts and categories

Besides the scanning of the actual posters, I needed the actual texts that were on the posters.

One text-file contains the lines of text for each poster. To keep things simple, LOESJE keyed in all the data. A typical entry looks like this:

```
N199312C Actual text, perhaps
sevaral lines

long [3\7\13]
```

The above means: The file N199312C.PNG is the actual poster containing this text, the year is 1993, the month is December (12) and this is the third poster (C) in that month. The poster falls in three categories: 3, 7 and 13.

The resulting typeset layout should observe the new- and empty lines in the files.

To convert the category numbers to actual category names there was another text file: `cat.txt`. This file looked something like this:

```
Alien
        9 Common
       10 Astrology
       11 Space
Future
       72 Common
       73 Dreams/ideals
       74 Plans
       75 2002
```

The above means: The main category 'Alien' contains the subcategories: Common (9), Astrology (10), and Space (11). The main category 'Future' contains the subcategories: Common (72), Dreams/ideals (73), Plans (74), and 2002 (75).

These files are fairly easy to scan with PERL. The scanning code is just a screen or two. After each text definition is scanned, a PERL object is constructed with the following attributes:

Year The year of the poster.

Month The month of the poster.

Categories An array containing the category numbers for this particular poster.

Text The actual text.

All the poster objects are put into a hash (a key-value pair array) where the key is the unique poster number (like N199312C).

After the scanning and building of the hash is complete the output-files are constructed.

4 Using different tools

I am a tool-guy. I use whatever tool that I know could do the job easily. The advantage here is obvious: the right tool for the right job gets the work done more quickly. There's also a disadvantage: I usually do not know the exact ins-and-outs of a tool. I know little of TeX, in fact the way TeX 'thinks' is definitely not my way. I see TeX as an enhanced M4 macro-processor, with weird syntax, nice output and unlimited possibilities.

Do not blame me for my limited vision of this powerful typesetting engine, it is just the way I work with it. My macros are not nice and I usually overlook powerful features, but they get the job done I hope. As I write macros (in any language, be it TeX or PERL) I experiment until the result is what I want. If the used tool can't do the job for me (usually because I am too stupid to find the right keywords) then I'll try another tool until the results are satisfactory.

The same story holds for PERL. If a take a look at the packages that come with PERL I am amazed by the possibilities. You can even write web-servers in PERL with just a few lines of code. I used PERL before to convert structured text documents to PDF and HTML with everything cross-linked and it is definitely a *very* powerful tool for doing system stuff like messing with files, directories and contents of files.

PERL and TeX have something in common: both are a bit weird, though PERL looks more like a conventional programming language to me. To achieve things in both tools, you can use several mechanisms and language constructs. This is better accepted in the PERL world than in the TeX world. I sometimes overhear conversations about TeX where people are trying to convince each other that their way is the best way to do it. I do not believe in such a concept. The best way to achieve things is the way that generates the most fun and gets the job done.

5 PERLing it away

This section should definitely not be read by any advanced TeX user and specifically any ConTeXt user. That's because they would claim that all this structurizing I did could easily be done from within ConTeXt. Okay, I admit that is very true, and ConTeXt does support a lot of usable stuff for me. The only problem is that I don't know them well enough.

Using PERL to scan the files was easy. Generating the output however was more difficult.

I first needed to know what kind of browsing these LOESJE guys would need. They wanted two things:

1. Chronological. You can browse through the poster-texts sorted on date. Below should be a button-bar with the available years, and above

that a button-bar with the months in that year. Each poster-text will be included once.

2. Categorial. It is similar to the chronological but organized by main- and subcategories.

So I decided to use a section/subsection mechanism as found in LaTeX.

A sample of the output:

```
\\YEAR{1993}
\\MONTH{January}
\startposter{N199312C}
Hi there, this is some poster text
\stopposter
```

The macros `\startposter` and `\stopposter` should do all the work (I'll come back to those later).

6 Using ConTEXt to do the layout

Many of you probably know ConTEXt as a very powerful program for creating interactive documents. If you don't believe me, try it for yourself. The trouble with ConTEXt is finding the right way to do it. There are usually several.

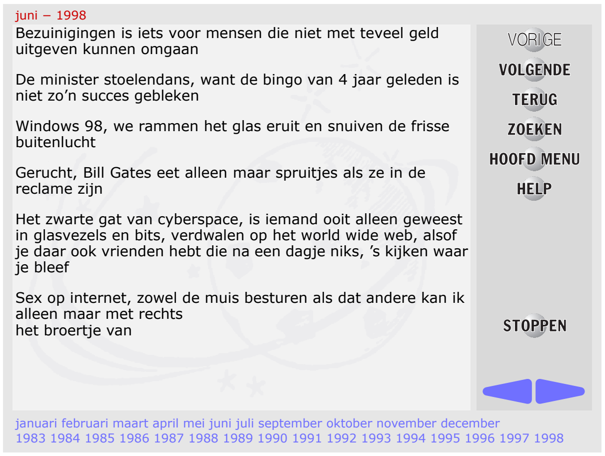
Almost all of the features found in the PDF specifications can be used. In some aspects ConTEXt defines more functionality than PDF has to offer. The whole concept behind ConTEXt is well thought-out and Hans Hagen is a true wizard when it comes to functionality and completeness. If you have a nice generic package or add-on, Hans is willing to integrate it in ConTEXt given the time. Modularity in ConTEXt is something weird, because the package is large and monolithic. In fact the basic services in ConTEXt are rather limited when it comes to 'I want to write an article'. But once you get the hang of it you discover that customizing things is a breeze, compared to whatever I've ever encountered in TEX miracle land. You do not need to know a lot about TEX (which is definitely a big plus) and it usually works the way you expect. And if you're not certain about the correctness of the output, you simply turn on the visual debugger,[1] and you can actually *see* where you forgot that extra percent sign, yielding that unwanted space.

6.1 Defining the layout

Take a look at figure 1. It is the basic layout. All the screens in the product are similar to this one, so I designed a basic layout to create this.

Defining the layout is simple. You first setup the papersize:

[1] Editor's note: This debugger is neat stuff—see the article by Hans Hagen in *TUGboat* 19(3) (September 1998), pp. 311*ff.*


juni – 1998

Bezuinigingen is iets voor mensen die niet met teveel geld uitgeven kunnen omgaan

De minister stoelendans, want de bingo van 4 jaar geleden is niet zo'n succes gebleken

Windows 98, we rammen het glas eruit en snuiven de frisse buitenlucht

Gerucht, Bill Gates eet alleen maar spruitjes als ze in de reclame zijn

Het zwarte gat van cyberspace, is iemand ooit alleen geweest in glasvezels en bits, verdwalen op het world wide web, alsof je daar ook vrienden hebt die na een dagje niks, 's kijken waar je bleef

Sex op internet, zowel de muis besturen als dat andere kan ik alleen maar met rechts
het broertje van

Figure 1: Basic layout

```
\setuppapersize
  [S6]
```

The S6 means: Screen based papersize. It is similar to A4, except that the width is 600pt and the height is 450pt. ConTEXt sets up margins and automatically calculates the text-area. For screen-based layout, there's one thing for sure: Whatever ConTEXt calculates, it is never what you want. (It works fine for paper-based output.)

So now let's setup the areas to be used:

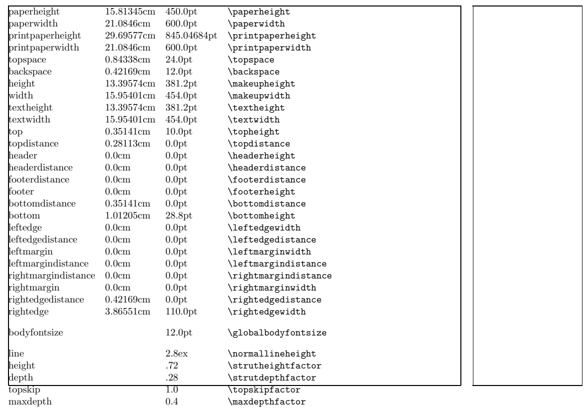
```
\setuplayout
  [topspace=24pt,
   header=0pt,
   footer=0pt,
   bottomdistance=10pt,
   bottom=28.8pt,
   topdistance=8pt,
   top=10pt,
   backspace=12pt,
   margin=0pt,
   edgedistance=12pt,
   rightedge=110pt,
   height=fit,
   width=fit]
```

What does this all mean?

- There is 24pt of space on the top, before the text-area begins.
- There is no header text (above the text) or footer text (below the text). These are usually used to put in page numbers of chapter headings. I don't need them for screen-layout.
- There is bottom-text below the text-area; its height is 28.8pt. There is also top-text above the text, height 10pt.
- There are no margins.

- The distance from the text-area to the edge is 12pt. The width of the right edge is 110pt.
- And finally we say: 'Dear ConTEXt, I do not know what the width and height of the text-area should be, so calculate them based on the given settings'.

That's about it. After that you can verify what you have done by saying: `\showframe`. ConTEXt then draws some frames where you defined them, so you can actually see what is going on. See figure 2.



```
paperheight         15.81345cm  450.0pt       \paperheight
paperwidth          21.0846cm   600.0pt       \paperwidth
printpaperheight    29.69577cm  845.04684pt   \printpaperheight
printpaperwidth     21.0846cm   600.0pt       \printpaperwidth
topspace            0.84338cm   24.0pt        \topspace
backspace           0.42169cm   12.0pt        \backspace
height              13.39574cm  381.2pt       \makeupheight
width               15.95401cm  454.0pt       \makeupwidth
textheight          13.39574cm  381.2pt       \textheight
textwidth           15.95401cm  454.0pt       \textwidth
top                 0.35141cm   10.0pt        \topheight
topdistance         0.28113cm   0.0pt         \topdistance
header              0.0cm       0.0pt         \headerheight
headerdistance      0.0cm       0.0pt         \headerdistance
footerdistance      0.0cm       0.0pt         \footerdistance
footer              0.0cm       0.0pt         \footerheight
bottomdistance      0.35141cm   0.0pt         \bottomdistance
bottom              1.01205cm   28.8pt        \bottomheight
leftedge            0.0cm       0.0pt         \leftedgewidth
leftedgedistance    0.0cm       0.0pt         \leftedgedistance
leftmargin          0.0cm       0.0pt         \leftmarginwidth
leftmargindistance  0.0cm       0.0pt         \leftmargindistance
rightmargindistance 0.0cm       0.0pt         \rightmargindistance
rightmargin         0.0cm       0.0pt         \rightmarginwidth
rightedgedistance   0.42169cm   0.0pt         \rightedgedistance
rightedge           3.86551cm   110.0pt       \rightedgewidth

bodyfontsize                    12.0pt        \globalbodyfontsize

line                            2.8ex         \normallineheight
height                          .72           \strutheightfactor
depth                           .28           \strutdepthfactor
topskip                         1.0           \topskipfactor
maxdepth                        0.4           \maxdepthfactor
```

Figure 2: Empty layout

The resulting PDF file still has a problem though: the layout is OK, but the page size is still A4. To overcome this you say:

```
\setupinteractionscreen
  [option=max,
   width=fit,
   height=fit]
```

This more or less means: Open the document in full-screen, and make the width and height fit to the calculated values.

6.2 Defining backgrounds

I have now defined a simple white page, so let's enhance it with background colors and graphics. If you do this the traditional way, by drawing something in the output routine, before you actually shipout the page you will have a problem: The resulting file will contain the graphics images on each and every page. That is not very efficient, since the background images are a total of 73Kb. If you multiply that by the number of pages (about 800) you will get a pdf file of 60 Megabytes. All of this while you know for sure that PDF supports something that's called *objects* to do the job properly.

Luckily you don't have to worry about that. You simply use and include as many figures as you like. If ConTEXt sees you've actually used the figure before, it won't include it again. It will simply creates a reference to the figure. So each figure is included only once.

First we define a background color for the entire page:

```
\setupbackground
  [page]
  [background=color,
   backgroundcolor=pagebackgroundcolor]
```

So what is `pagebackgroundcolor` you might ask. Well:

```
\definecolor
  [pagebackgroundcolor]
  [r=.9, g=.9, b=.9]
```

Easy ha? You can probably guess how to define CMYK colors. The right edge has a similar background color, but with a little different setup:

```
\setupbackgrounds
  [text][rightedge]
  [background=color,
   backgroundcolor=edgebackgroundcolor]
```

So now for the picture in the text-area. This is a bit more difficult, because this works with overlay techniques:

```
\defineoverlay
  [world]
  [{\externalfigure
    [world]
    [width=\overlaywidth,
      height=\overlayheight]}]

\setupbackgrounds
  [text][text]
  [background={color,world},
   backgroundcolor=textbackgroundcolor]
```

What's this? We are defining two backgrounds here, a color, and a picture that comes from the file `world.eps`. The order of specification defines the overlaying. I do not know if there are any limits here but knowing Hans there is probably no real limit to the number of backgrounds you can stack upon each other.

Also note the sizing of the image. The image is a bitmap converted to an .eps file (using CorelDraw) and has some transparency. You only need to get the aspect ratio of the .eps file correct, and ConTEXt automatically scales the image to the width and height of the text area.

6.3 Defining the right-edge navigation bar

The navigation bar on the right contains many buttons stacked above each other. The normal command to do this is `\button`. But since the buttons have backgrounds I needed to define the backgrounds first. The trick is this: You first define what you want to have, put it all in a box and then you say to ConTEXt:

```
\setuptexttexts
  [edge]
  [][\ButtonList]
```

This means: The content of the left edge is empty, and the right edge contains `\ButtonList`.

`\ButtonList` is defined as follows (some buttons are left out here to save some space):

```
\def\ButtonList{%
  \hbox to \rightedgewidth{\vbox to \vsize
    {\midaligned{\previousbutton}\par
     \midaligned{\nextbutton}\par
     ...
     \vfill
     \midaligned{\stopbutton}\par
     \vfill
     \midaligned{\PrevNextBar}\par
     \vss}}}
```

To define a button:

```
\def\previousbutton{%
  \button
    [frame=off,
      width=104bp,
      height=25bp,
      background=previous]
      {}
      [previouspage]}
```

There is no frame around the button, because I supplied a picture. The button contains no text (the empty braces `{}`), and when someone presses the button, the previous page should be displayed. All buttons are created this way.

Now there is one little speciality here: the `\PrevNextBar` command. When a series of texts is displayed, and they don't fit on a single screen, a button automatically appears indicating that there is more to see in this series. Now I wanted those buttons to be smart: On the first page you only see a button to go to the next page, and in the last page of a series you only see a button to go to the previous page. The pages in between (if any) have both buttons enabled.

Again ConTEXt helps me out here. There is something called sub-page numbering. I setup sub-page numbering by series, and when placing the buttons I ask ConTEXt how many pages there are in this series. Based on that, and the current sub-page number, I include the correct buttons:

```
\def\PrevNextBar%
  {\ifnum\nofsubpages>1
     % all right, we need some buttons
     % here
     \ifnum\subpageno=1
       % This is the first page so only
       % include the right button
       \framed
         [frame=off,
           width=\arrowwidth,
           height=25bp]{}\hss
         \arrowrightbutton%
     \else\ifnum\subpageno=\nofsubpages
       % This is the last page, so only
       % include the left button
       \arrowleftbutton\hss
     \else
       % We are somewhere in between
       % include both buttons
       \arrowleftbutton\hss
       \arrowrightbutton%
     \fi\fi
   \else
     % No sub pages, so no buttons
     \framed
       [frame=off,
         height=25bp,
         width=\rightedgewidth]
       {}
   \fi}
```

Now that's a nice trick, isn't it?

6.4 Defining the bottom navagation bar

The bottom navigation bar is a bit more difficult, since it contains data from the actual data-file itself. Since all years and months are placed in lists, I can easily extract that information and put it anywhere I want. I only provide the details here, the actual commands to setup texts in the bottom area are similar to the right-edge button bar. It consists of two vboxes, a box for the months and one for the years.

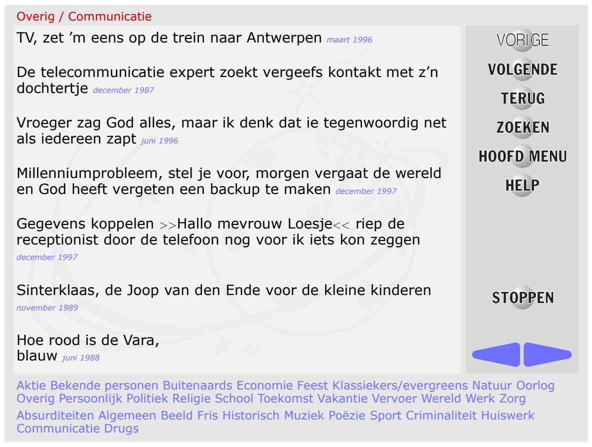
```
\def\YearMonthList{%
  \vbox to \bottomheight{%
    \vbox{%
      \placelist
        [MONTH]
        [variant=none,
         criterium=YEAR,
         command=\SIMPLE,
          before=\strut]
        \vss}%
    \vss
    \vbox{%
      \placelist
        [YEAR]
        [variant=none,
         criterium=all,
         command=\SIMPLE,
         before=\strut]
        \vss}}}

\def\SIMPLE#1#2#3{#2}
```

At first this didn't do the job, because all years were on a separate line. I guess the `\placelist` results in a lot of vboxes, and these are stacked vertically by TEX. By adding the `\strut` the problem was over. This is probably not the preferred solution, but I can't write an email to Hans every time I am in trouble.

As you can see I didn't provide any commands for specifying the interaction and high-lighting. This is what ConTEXt automatically does for me.

7 Results and some samples

Take a look at the figures. Figure 3 shows the main screen of the categories.


Overig / Communicatie

TV, zet 'm eens op de trein naar Antwerpen maart 1996

De telecommunicatie expert zoekt vergeefs kontakt met z'n dochtertje december 1987

Vroeger zag God alles, maar ik denk dat ie tegenwoordig net als iedereen zapt juni 1996

Millenniumprobleem, stel je voor, morgen vergaat de wereld en God heeft vergeten een backup te maken december 1997

Gegevens koppelen >>Hallo mevrouw Loesje<< riep de receptionist door de telefoon nog voor ik iets kon zeggen december 1997

Sinterklaas, de Joop van den Ende voor de kleine kinderen november 1989

Hoe rood is de Vara, blauw juni 1988

Aktie Bekende personen Buitenaards Economie Feest Klassiekers/evergreens Natuur Oorlog Overig Persoonlijk Politiek Religie School Toekomst Vakantie Vervoer Wereld Werk Zorg Absurditeiten Algemeen Beeld Fris Historisch Muziek Poëzie Sport Criminaliteit Huiswerk Communicatie Drugs

Figure 3: A category screen

Figure 4 is an actual poster.

8 Conclusion

Well, what can I say? ConTEXt saved me a lot of time, and the remaining time creating the product was fun to spend. I have only covered some basic issues here; there are several other commands needed to get things going.

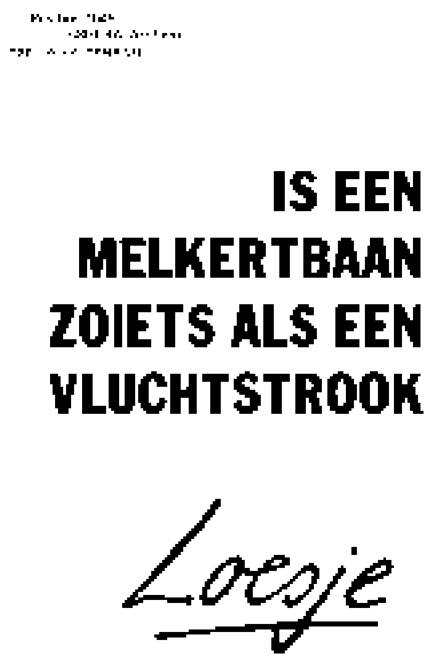


Figure 4: An actual poster

Using the example files provided by ConTEXt gave me a good idea of the possibilities. Though the example presentations do not look like this product, they serve as good examples of what is possible.

I liked the tooling very much. It is easy to generate TEX documents from PERL, and ConTEXt is pretty relaxed in using such documents.

You have probably discovered that the included graphics are in Dutch. Most of the words are what you expect them to be, so I expect this is not a big problem.

◇ Gilbert van den Dobbelsteen
Papaverstraat 130
7514 XH Enschede
Netherlands
`gilbert@login-bv.com`

NetBibTEXing

Robert Tolksdorf

1 Introduction

BibTEX is the format of choice for cataloging and referring to literature with currently highest importance for scientific references. It defines a standard format for keeping meta-information on published material, a language to process these, and an implementation of the processor, the `bibtex` program.

Usually, the user collects bibliographic information in local BibTEX databases manually. With the widespread availability of the Internet, more and more bibliographic collections have been made available online via the Web. With the immense growth of the number of entries available, the need for services that help in locating reference information has increased.

An example is the "The Collection of Computer Science Bibliographies" at `http://liinwww.ira.uka.de/bibliography`. It collects over 1200 bibliographies that contain more than 940000 references and provides a search service on this data. The collections are well maintained by their respective authors and show a high timeliness.

In this article, we describe the design and implementation of NetBibTEX, a system that uses such a service online to retrieve bibliographic references based on a special kind of citations in a LATEX document.

2 Overview

Figure 1 gives an overview on the files and processors involved. NetBibTEX contains a style-file that allows the inclusion of *netcitations* in the document source. Similar to a normal `\cite`, they require a bibliographic key, but also describe the reference by keywords for names fields, such as

```
\netcite{robert:lcs}{title=Coordination
 Laura,author=Robert Tolksdorf,year=1998}
```

The author will later interactively select a reference retrieved from the net based on this description — the *netreference*. For each netcitation, an entry in a special `.nbb`-file is generated that contains the information marked up with XML:

```
<netbibqueries value="Version 1.0">
<bibquery value="robert:lcs">
<title value="Coordination Laura"/>
<author value="Robert Tolksdorf"/>
<year value="1998"/>
</bibquery>
</netbibqueries>
```

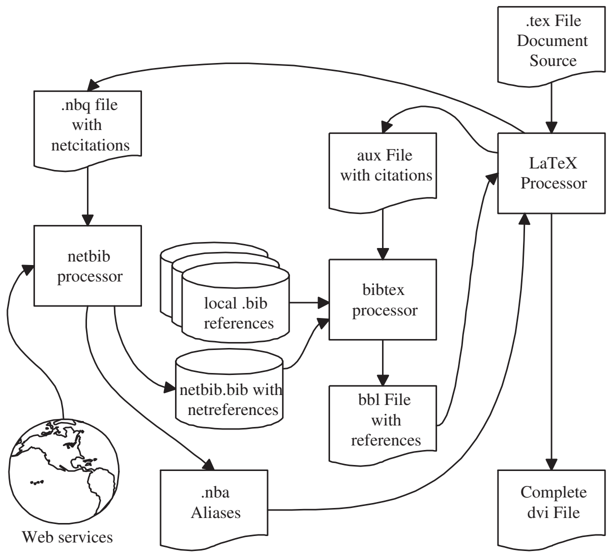


Figure 1: Overview of NetBibTEX

The choice of XML-format is motivated by the availability of the language WebL, a scripting language that allows easy extraction of information from XML/HTML marked up documents (**?**). We use it to implement the `netbib` processor. It performs a query to Web-services that collect bibliographic references for each netcitation. From the answers retrieved, it extracts the netreferences and asks the user to select one as shown in figure 2.

Each selected entry is written into `netbib.bib`. In addition, the key used in the netcitation has to be mapped to the actual key in the netreference. NetBibTEX provides a document style `bibalias` that allows the definition of aliases for bibliographic keys. `netbib` generates an `.nba` file that contains the appropriate definitions:

```
% Generated BibTeX key aliases by netbib
\bibalias{robert:lcs}{SCP::Tolksdorf1998}
```

With these mechanisms the queries from netcitations are matched by netreferences. These are stored in the generated bibliography and their keys are aliased with the keys used in the netcitations. Three LATEX and one BibTEX runs are needed to produce the final, complete document.

```
grunge tolk 3 (~/bibalias): webl netbib.webl netbibtest

@Article{SCP::Tolksdorf1998,
  title =       "Laura---{A} service-based coordination language",
  author =      "Robert Tolksdorf",
  pages =       "359--381",
  journal =     "Science of Computer Programming",
  month =       jul,
  year =        "1998",
  volume =      "31",
  number =      "2--3",
  references =   "\cite{PPOPP::AghaC1993} \cite{ACMTCS::BirmanSS1991}
                 \cite{TOPLAS::BowmanDP1993}
                 \cite{ACMTCS::CarrieroG1986}
                 \cite{CACM::GelernterC1992}",
}

Accept this entry for citation "robert:lcs"?y
```

Figure 2: Selecting a reference found in the net

3 User interface

The user interface of NetBibTeX is very small— there is one LaTeX macro for netcitations and the `netbib` program.

3.1 Netcitations

A document using NetBibTeX has to include the package `netbib` with `\usepackage{netbib}`. It provides the macro `\netcite`, includes generated aliases at the start of the document and finalizes the generation of the `.nbq` at the end of the document. In addition, it changes the behavior of `\bibliography` to include the generated `netbib.bib`.

To use a netcitation in a document, one uses `\netcite{`⟨*bibkey*⟩`}{`⟨*query*⟩`}` as in the example above. ⟨*bibkey*⟩ is a bibliographic key for a reference, identical to the ones used with `\cite`. ⟨*query*⟩ describes the reference by a comma-separated list of keyword queries to fields. Note that this format is defined by NetBibTeX and is mapped to specific queries for Web-services by the `netbib` program. The fields defined are:

- `author`: The author of the cited work
- `title`: The title of the cited work
- `year`: The year of publication of the cited work
- `key`: The whole citation

In addition, two flags can be used in the query:

- `word`: Consider only complete words exactly
- `case`: Distinguish upper- and lowercase

Note the condition in this description: If the Web-service used by `netbib` does not provide the respective searching options, then the flags are ignored.

NetBibTeX will try to construct a "good" query to various services and favors conjunction of given keywords for the fields in order to narrow the set of possible matches as much as possible. The system could be extended to further control that behavior.

For each service used by NetBibTeX, a special routine has to be programmed that maps the netquery into a specific query to the respective Web service using the specific query syntax there. In the initial `netbib.webl` script, we demonstrate this for three services in the functions `queryGibbens`, `queryPPA`, and `queryCSBibColl`.

The extraction of BibTeX entries from the results of the queries is also dependent on the services used. In the three cases implemented, we pose queries that result in a single HTML page with a list of possible matches. It contains references in BibTeX syntax enclosed by the `<PRE>` tag in all three cases. The WebL function `Elem` returns a set of page fragments, each being one of the preformatted BibTeX entries.

Depending on the services used, the extraction could be implemented in a different manner. One could also program conversation routines if citation services return other formats than BibTeX.

Administrators of bibliographic collections can contribute to NetBibTeX by programming a respective query and extraction routine, or by documenting the syntax of their queries and the format of the output in detail. E-mails with information on extending NetBibTeX with further services are highly appreciated by the author.

3.2 Interactive selection of netreferences

After writing out the queries for netcitations in the `.nbq` file, the `netbib` processor can perform a search on the Web for matching references. It is implemented in the scripting language *WebL*, which is an interpreter written in Java.

We have chosen Java as the underlying execution mechanism to implement platform independence of `netbib`. The current drawbacks in execution time are not relevant for `netbib`, as its speed is dominated by the external Web services that look for references and by the network latency.

In order to use WebL, you need a Java virtual machine and the WebL interpreter available free (including sources) at `http://www.compaq.com/WebL`. Follow the respective instructions for installation of WebL.

Assuming that there is some script `webl` installed that starts Java with the main class of `webl.jar`, the NetBibTeX processor is started for a document ⟨*document*⟩`.tex` with
`webl netbib.webl` ⟨*document*⟩

The program starts to extract the netcitations and searches for netreferences. For each one, the user is asked as in figure 2 whether to accept it.

The GUI shown is very clumsy and not very convenient to use. We will put an extended version with a graphical interface for the selection of references at the Web site mentioned below. Its implementation involves specific techniques to access Java-classes from WebL that are of no interest here.

The retrieval of netreferences is rather slow and the selection of a matching one can be very tedious if the query is not very precise and the service offered a long list of references.

In order to avoid unnecessary queries, the `netbib`-style put a

tag `<known/>` into each `bibquery` for which a netreference has already been retrieved. This is detected by testing whether the key used in the netcitation is an alias for a netreference. By using the option `-a` for the `netbib` program, this tag is ignored and all netcitations are (re-)processed.

4 Outlook

NetBibTeX is both expandable and dependent with respect to Web services that offer to search bibliographies and output results in BibTeX format. The implementation shown in the appendix might well lead to unpredictable results due to changes in URLs or forms. At `http://www.cs.tu-berlin.de/~tolk/netbib` you can find the homepage of NetBibTeX that includes the latest versions of the system.

◇ Robert Tolksdorf
Technische Universität Berlin
Fachbereich 13, Informatik
FLP, FR 6-10
Franklinstr. 28/29
D-10587 Berlin
Germany
`tolk@cs.tu-berlin.de`
`http://www.cs.tu-berlin.de/~tolk`

A The Implementation

NetBibTeX consists of the two LaTeX stylefiles `bibalias.sty` and `netbib.sty`, and the WebL Script `netbib.webl` that are documented below.

A.1 `bibalias.sty`

First, we introduce ourselves.

```
\ProvidesPackage{bibalias}
```

`\bibalias` For a key k_1 which is an alias for a key k_2, we define a label `ba@`k_1 that expands to k_2.

```
\newcommand{\bibalias}[2]{\@newl@bel{ba}{#1}{#2}}
```

`\@citex` Citations are expanded into the respective labels in the `\@citex` macro. The individual references are extracted from the comma-separated list in the second parameter and processed as `@citeb`. The first lines of the macro are copied directly from `latex.ltx`.

```
\def\@citex[#1]#2{%
  \let\@citea\@empty \@cite{\@for\@citeb:=#2\do
    {\@citea\def\@citea{,\penalty\@m\ }%
     \edef\@citeb{\expandafter\@firstofone\@citeb\@empty}%
```

Here we test whether the key is an alias for another one.

```
      \@ifundefined{ba@\@citeb}{}%
```

Yes, it is an alias. We replace the content of `\@citeb` with the alias.

```
         {\typeout{\@citeb\space is an alias for \@nameuse{ba@\@citeb}}%
         \global\edef\@citeb{\@nameuse{ba@\@citeb}}}%
```

Note that we do not support aliased aliases here. The remainder of `@citex` is again a copy of the original LaTeX-code.

```
       \if@filesw\immediate\write\@auxout{\string\citation{\@citeb}}\fi
      \@ifundefined{b@\@citeb}{\mbox{\reset@font\bfseries ?}%
        \G@refundefinedtrue
        \@latex@warning
          {Citation `\@citeb' on page \thepage \space undefined}}%
        {\hbox{\csname b@\@citeb\endcsname}}}}{#1}}
```

A.2 netbib.sty

First, we introduce ourselves.

```
  \ProvidesPackage{netbib}
```

We depend on bibalias for aliasing the keys of netcitations to the actual ones found in the net and `keyval` from the standard LaTeXgraphics bundle for dealing keyword-value lists.

```
  \RequirePackage{bibalias}
  \RequirePackage{keyval}
```

We now define the allowed set of keywords and their processing. When `\setkey` parses a list, it handles the keywords defined here and processes them by the commands in the third argument of `\define@key`. For each keyword, we write out a tag.

```
  \define@key{netbib}{author}{\nb@writevaluetag{author}{#1}}
  \define@key{netbib}{title}{\nb@writevaluetag{title}{#1}}
  \define@key{netbib}{year}{\nb@writevaluetag{year}{#1}}
  \define@key{netbib}{key}{\nb@writevaluetag{key}{#1}}
  \define@key{netbib}{word}[true]{nb@writetag{word}}
  \define@key{netbib}{case}[true]{\nb@writetag{case}}
```

`\nb@queryfile` refers to the `.nbq` file that contains the netcitations in XML markup.

```
  \newwrite\nb@queryfile
  \immediate\openout\nb@queryfile=\jobname.nbq
```

The following four handy macros write out XML tags.

```
  \def\nb@writetag#1{\protected@write\nb@queryfile{}{\string<#1/>}}%
  \def\nb@writevaluetag#1#2{\protected@write\nb@queryfile{}{\string<#1 value="#2"/>}}%
  \def\nb@openvaluetag#1#2{\protected@write\nb@queryfile{}{\string<#1 value="#2">}}%
  \def\nb@closetag#1{\protected@write\nb@queryfile{}{\string</#1>}}%
```

The XML startsymbol for our `.nbq` files is `<netbibqueries>`, thus we generate such a tag immediately and close it at the end of the document.

```
  \nb@openvaluetag{netbibqueries}{Version 1.0}
  \AtEndDocument{\nb@closetag{netbibqueries}}
```

`\netcite` `\netcite` is used with a key in the first argument and a keyword-value list in the second argument. `\setkeys` processes the keyword list and thus generates several tags. We encapsulate them with a tag-pair `<bibquery value="`*key*`">`. If there is already an alias for the citation key, then we generade a tag `<known/>` in the `.nbq` file to avoid unnecessary network queries. The final `\cite` will later cite an alias to a netreference, or generate a LaTeX message.

```
  \def\netcite#1#2{%
  \nb@openvaluetag{bibquery}{#1}%
  \setkeys{netbib}{#2}%
  \@ifundefined{ba@#1}{}{\nb@writetag{known}}
  \nb@closetag{bibquery}%
  \cite{#1}}
```

`\bibliography` `\bibliography` has to include the generated `netbib.bib` for the netcitations. We redefine it to extend its argument appropriately and then leave the work to the original macro that we remember in `\oldbibliography`.

```
\let\oldbibliography\bibliography
\def\bibliography#1{%
  \ifx#1\relax \oldbibliography{netbib,#1}
  \else        \oldbibliography{netbib}
  \fi}
```

`netbib` generates an `.nba` file that contains the alias definitions for the netcitations. It has to be read at the beginning of the document.

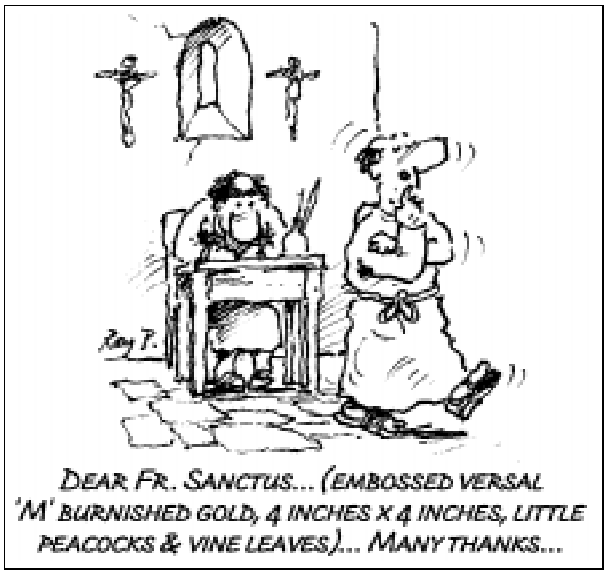
```
\AtBeginDocument{\@input{\jobname.nba}}
```

A.3 `netbib.webl`

```
// import some modules
import Url, Str, Files;

// if elem!=nil return the value attribute, nil otherwise
var valueOrNil = fun(elem)
  if (elem!=nil and Size(elem)>0) then return elem[0].value else return nil end;
end;

// if elem is empty, return nul
var trueOrNil = fun(elem)
  if (Size(elem)>0) then return elem else return nil end;
end;

// This service knows named fields - we construct the respective and-separated query
var queryCSBibColl = fun(author,title,year,key,word,case)
  var andString="", query="", case="off", partial="on";
  if (author!=nil) then query="au="+author;                andString=" and " end;
  if (title!=nil)  then query=query+andString+"ti="+title; andString=" and " end;
  if (year!=nil)   then query=query+andString+"yr=="+year; andString=" and " end;
  if (key!=nil)    then query=query+andString+"text="+key; andString=" and " end;
  // the advanced query that we use here does not support case and word
  var result=PostURL("http://liinwww.ira.uka.de/waisbib",
    [. database="local/bibliography", convert="bibtex", directget="1",
       sortmode="score", text=query, maxhits="170" .]);
  return Elem(result,"pre");
end;

// This service uses only keywords for the search
var queryGibbens = fun(author,title,year,key,word,case)
  var query="", type="substr";
  if (author!=nil) then query=author+" "      end;
  if (title!=nil)  then query=query+title+" " end;
  if (year!=nil)   then query=query+year+" "  end;
  if (key!=nil)    then query=query+key+" "   end;
  // case handling is unspecified by the service, words are handled
  if (word!=nil)   then type="exact" end;
  var result=PostURL("http://www.statslab.cam.ac.uk/cgi-bin/bibsearch.pl",
     [. header="~richard/misc/biblio/header", footer="~richard/misc/biblio/footer",
        files="~richard/misc/biblio/rjg.bib", term=query, field="all", type=type .]);
  return Elem(result,"pre");
end;

// This service uses named fields and expects a query starting with "find"
var queryPPA = fun(author,title,year,key,word,case)
  var query="find ", andString="";
  if (author!=nil) then query=query+"author "+author+" ";     andString=" and " end;
  if (title!=nil)  then query=query+andString+"title "+title; andString=" and " end;
  if (year!=nil)   then query=query+andString+year;           andString=" and " end;
  if (key!=nil)    then query=query+andString+key;            andString=" and " end;
  // word is ignored by netbib - we do not construct wildcards, case is ignored by the service
  var result=PostURL("http://wwwslap.cern.ch/cgi-bin/collective/bibsearch2.pl",
    [. query=query, output="BibTeX" .]);
  return Elem(result,"pre");
end;

// entries is a pieceset with each piece containing a bibtex record. select shows each to
// the user and prompts for a selection. This one is returned, or nil if nothing was selected
var select = fun(entries,key)
  every entry in entries do
    every t in PCData(entry) do  Print(Text(t)) end;       // write out the record
    Print("\nAccept this entry for citation \""+key+"\"?");
    var answer=ReadLn();                                   // ask for a selection
    if (answer=="Y" or answer=="y") then return(entry) end // return, if this one is accepted
  end;
  return nil;  // if no entry was selected, return nil
end;

PrintLn("This is netbib, Version 1.0");

// Process the command line
var fileName;
var queryAll=(ARGS[1]=="-a");    // check for -a option
if queryAll then fileName=ARGS[2] else fileName=ARGS[1] end;

// The names of the generated bibliography and aliases files
var entriesFile = "netbib.bib", aliasesFile = fileName+".nba", queryFile   = fileName+".nbq";

// Write out information to them
if (queryAll) then
  Files_SaveToFile(entriesFile,"% Generated BibTeX entries by netbib\n");
  Files_SaveToFile(aliasesFile,"% Generated BibTeX key aliases by netbib\n");
else
  Files_AppendToFile(entriesFile,"% Generated BibTeX entries by netbib\n");
  Files_AppendToFile(aliasesFile,"% Generated BibTeX key aliases by netbib\n");
end;

// Read in the query file and extract all netcitations as a pieceset
var queries = Elem(Files_LoadFromFile(queryFile,"text/xml"),"bibquery");

// The list of wrapper methods to query Web-services
var services = [queryGibbens, queryPPA, queryCSBibColl];

var selection, entries;
// process all netcitations
every query in queries do
  // If it has no matching netreference yet, or everything is reprocessed
  if (trueOrNil(Elem(query,"known"))==nil or queryAll) then
    selection= nil;
    PrintLn("Searching netreference for "+query.value);
    // Query services until a netreference is selected
    while (selection==nil) and (Size(services)>0) do
      PrintLn("Quering a service");
      entries = (First(services))(valueOrNil(Elem(query,"author")),
                       valueOrNil(Elem(query,"title")),
                       valueOrNil(Elem(query,"year")),valueOrNil(Elem(query,"key")),
                       trueOrNil(Elem(query,"word")), trueOrNil(Elem(query,"case")));
      services=Rest(services);
      selection = select(entries,query.value);
      if (selection!=nil) then
        // extract BibTeX key from selection
        var bibkey=Str_Match(Text(PCData(selection)[0]),‘^(\s|\S)*@\S*\{\s*(.*),(\s|\S)*‘)[2];
        // alias the key used in \netcite to the one from the net
        Files_AppendToFile(aliasesFile, ‘\bibalias{‘+query.value+‘}{‘+bibkey+‘}‘+"\n");
        // writeout entry into netbib.bib
        every t in PCData(selection) do Files_AppendToFile(entriesFile,Text(t)) end;
      end;
    end;
  end;
end
```

Cartoon

by Roy Preston


Roy P.
DEAR FR. SANCTUS... (EMBOSSED VERSAL 'M' BURNISHED GOLD, 4 INCHES X 4 INCHES, LITTLE PEACOCKS & VINE LEAVES)... MANY THANKS...

Hints and Tricks

‘Hey — it works!’

Jeremy Gibbons

Welcome to *‘Hey — it works!’*, a column devoted to (LA)TEX and METAFONT tips, tricks and techniques. Any short and elegant TEX-related items are warmly received.

In this issue, as usual, we have three articles. One is my own, and provides a macro for margin notes that you can switch on and off. The second is by Andreas Scherer, and shows how to draw smooth graphs using METAPOST’s `graph` package. The third is by Ramón Casares, and demonstrates how to disable TEX’s rule for deciding whether a ‘`.`’ ends a sentence.

Last issue (Vol. 19, No. 4) we had an article by Christina Thiele showing how to produce ornamental rules out of ordinary symbols, using `\cleaders`. The final paragraph demonstrated how to alternate two symbols, but to get an odd number of symbols properly laid out involved switching to `\leaders` instead of `\cleaders`, with the result that the rule is no longer centred within the requested width:

‘×÷×÷×÷×÷×   ’

Barbara Beeton responded to point out that it is not hard to recentre the rule: first trim its width to the actual printed width, and then ‘manually’ centre this trimmed rule within the requested width:

```
\def\bordertwo#1#2#3{{%
  \setbox1=\hbox{#1}%
  \setbox2=\hbox{#1#2}%
  \dimen0=#3\advance\dimen0 by -\wd1
  \divide\dimen0 by \wd2
  \multiply\dimen0 by \wd2
  \leavevmode
  \hbox to #3{\hfil#1\hbox to \dimen0
    {\leaders\hbox{#2#1}\hfil}\hfil}%
}}
```

This generates

‘ ×÷×÷×÷×÷× ’

instead.

◇ Jeremy Gibbons
CMS, Oxford Brookes University
Gipsy Lane, Headington
Oxford OX3 0BP, UK
jgibbons@brookes.ac.uk
http://www.brookes.ac.uk/~p0071749/

1 Switchable marginal notes

I often find it convenient, when working on the draft of an article, to annotate it in the margin with reminders of facts to check, corrections to make and so on. Of course, this is what LATEX’s `\marginpar` macro is for. However, I would also like to be able to switch off the annotations, for example when I am distributing the draft article to an audience for whom the annotations are inappropriate or irrelevant. I don’t want to have to edit the document to remove the annotations one by one; that’s just too much trouble. To solve this problem I wrote a simple macro for ‘switchable marginal notes’.

The following definitions should be put into a style file, say `margnote.sty`:

```
\newif\ifmarginnotes \marginnotestrue

\def\marginnotestyle
  {\scriptsize\itshape\raggedright}

\def\marginnote#1{%
  \@bsphack
  \ifmarginnotes
    \marginpar{\marginnotestyle#1}%
  \fi
  \@esphack}
```

Then marginal notes can be used by including

```
\usepackage{margnote}
```

in the document preamble.

The first line defines the marginal note switch. Marginal notes are turned on by default, but they can be turned off simply by saying

```
\marginnotesfalse
```

after the `\usepackage` declaration. (They can even be turned off and on mid-document.)

The second section specifies the style of marginal notes; by default, they are in a smaller size, italic, and set ragged right, but this can be changed by using `\renewcommand`.

The remainder of the file defines the marginal note macro itself. This takes a single argument, the text of the note, and sets it using `\marginpar` if marginal notes are turned on. For example,

```
\marginnote{This is a marginal note.}
```

produces the note in the margin here. The macros `\@bsphack` and `\@esphack` are internal to LATEX; they ensure that an entity like a marginal note or label definition does not introduce any extra space into a paragraph, independently of whether or not it is attached to a word.

This is a marginal note.

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

2 Smoothing augmented paths in METAPOST

The user manual of the METAPOST `graph` package states that neighbouring points of a path created with the `augment` macro are connected by straight line segments. Depending on the application, it may be more suitable to draw a smooth curve through the set of points on the path, using the '`..`' operator. This can be achieved easily.

Let the input be an external data file `hiw.data` containing several pairs of coordinates, each pair on a separate line:

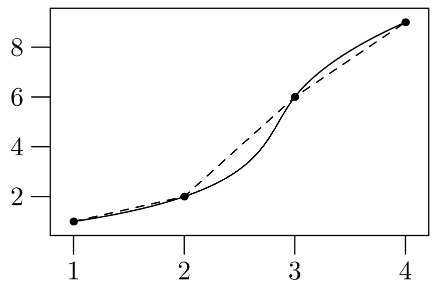
```
1 1
2 2
3 6
4 9
```

The following METAPOST code creates a (jagged) path by calling `augment` as the third argument to the `gdata` routine:

```
input graph;

beginfig( 1 );
  draw begingraph( 5cm, 3cm );
    path p;

    gdata( "hiw.data", c,
        augment.p( c1, c2 ); );

    gdraw p dashed evenly;
    gdraw (point 0 of p
      for i = 1 upto length p:
        .. point i of p
      endfor);

    pickup pencircle scaled 3pt;
    for i = 0 upto length p:
      gdraw point i of p;
    endfor;
    pickup defaultpen;

  endgraph;
endfig;
end.
```

This path is `gdraw`n the first time as a dashed line, depicting the default behaviour of `augment`. The "Hey, it works!" effect is achieved in the next four lines by `gdraw`ing a (temporary) smooth version of the same path. This is done directly as an argument to `gdraw`; no new variables are needed. Note how this is done in a simple `for` loop running over the points of the paths, applying the '`..`' operator in between.

Together with the control points displayed as heavy dots, the result of this code is shown in the following picture.



METAPOST automatically scales the x- and y-axes, adds a frame (whose size was set in the `begingraph` command), and attaches tick marks and labels.

METAPOST's `graph` package will not generate cyclic paths, but nevertheless a similar approach can be used to draw a smooth version of a cyclic polygon:

```
draw (for i = 0 upto (length p - 1):
    point i of p ..
  endfor cycle);
```

◇ Andreas Scherer
Rochusstraße 22–24
52062 Aachen, Germany
andreas.scherer@pobox.com

3 Every point a period

The rule used by TEX to decide whether a point is a period ending a sentence (so it will stretch the following space) or is just indicating an abbreviation is, for a simple mind like mine, too complicated. And it fails more frequently than expected when my text is full of ugly acronyms. So I have devised an alternative scheme.

Basically the idea is that every point is a period ending a sentence, so when I want to use a point in any other circumstance I have to protect the space that follows it, if any. If I want this space to be breakable then the solution is to write a backslash between the point and the space, that is '`.\␣`'. If, on the other hand, I want this space to be unbreakable then the solution is to write a tilde between the point and the space, that is '`.~␣`'. Easy, isn't it?

The code to achieve this is as follows:

```
\count255=`A
\loop
  \sfcode\count255=1000
  \ifnum\count255<`Z\advance\count255 1
\repeat
\def~{\nobreak\ \ignorespaces}
```

Note that I have appended a `\ignorespaces` to the tie mark definition (so in fact a space after a tilde is ignored).

◇ Ramón Casares
Telefónica de España
r.casares@computer.org

Abstracts

Les Cahiers GUTenberg Contents of Issue 31 (December 1998) and Issue 32 (May 1999)

Issue 31

This issue of the *Cahiers* was set in *Apolline*, by Jean-François Porchez. For more information about font variations across the *Cahiers* series, see the end of this column.

Jacques André, Éditorial; pp. 3–4

The editor begins by stating that each issue of the *Cahiers* has its own look. That of the current one has been affected by the following keywords: "Delay", "Regular" [i.e., non-thematic], "Corrections", and "Technique". Each factor is then examined in turn, with each benefitting from Jacques' characteristic wry humour.

It should be noted that the article by Esperet and Girou is complemented by an article by André and Girou which appeared in *TUGboat* 20,1 (1999), pages 8–14.

Philippe Esperet and Denis Girou, Coloriage du pavage dit "de Truchet" [Coloring of "Truchet tiles"]; pp. 5–18

Three years ago, an algorithmic problem on tiling of a plane was set as a contest puzzle. After presenting various aspects to the puzzle, we give the main answers received. The winner was Rouben Ter Minassian. [Author's abstract]

The abstract does not allude to the fact that there are a good number of colour images of various Truchet 'tiles' included! Solutions showing three different approaches are provided: one based on PostScript, one on METAPOST, and one using PSTricks. As with all articles in the *Cahiers*, this one can be had as a downloadable file from the GUTenberg website (see new address at the end of this column).

Denis Roegel, Anatomie d'une macro [Anatomy of a macro]; pp. 19–27

This article provides a detailed explanation of a macro to calculate prime numbers. It also provides us with an opportunity to highlight some lesser-known TEXnical concepts.

[Author's abstract]

While the explanations may be in French, the macro in its entirety is just a collection of the same old English control sequences we've all come to ... look sideways at ... as we move along to something else ... `;-)`). The blow-by-blow account of what's happening is what makes this particular presentation quite interesting ...

Daniel Taupin, `ltx2rtf` : envoi de documents LATEX aux usagers de Word [`ltx2rtf`: sending LATEX documents to Word users]; pp. 28–37

The `ltx2rtf` compiler translates LATEX source files into RTF, a format available in many text editors, notably Microsoft Word. Originally written by Fernando Dorner and Andreas Granzer, students in Vienna (Austria), the initial version can be found on CTAN under the name `latex2rtf`. During the period 1997–98, we corrected and adapted the original version to run with LATEX 2_ε, under the name `ltx2rtf`. The distribution is mainly intended for use under MS-DOS Win95 and Win 3.11, but the program, written in standard C, can be compiled on any UNIX system with a CGG compiler.

[Author's abstract, with corrections]

The author's conclusion is worth noting:

> Just as it is not the purpose of dvips to allow compositors to replace LATEX with PostScript, `ltx2rtf` is not intended to have people abandon LATEX in favour of Word. Its sole purpose is to facilitate the transmission of properly formatted documents to people whose only viewing and/or printing tools are those provided by Microsoft. By doing so, it greatly expands the potential group of recipients of files originating in LATEX.

Sophie Brissaud, La lecture angoissée ou la mort du correcteur [Painful reading or, death of the proofreader]; pp. 38–44

This paper was first published at the ATypI conference at Lyons, in October 1998. The author reminds us that proofreading ought to be done only by professionals. She claims that it would be a pity if proofreaders were to disappear.

[Author's abstract]

The article is followed by a response from Jacques André, editor of the *Cahiers*.

Jacques André, Petite histoire des signes de correction typographique [A brief history of proofreaders' marks]; pp. 45–59

The history of the most important proofreaders' marks is shown. These marks are as old as printing. This fact is a sure indication that typographical quality has always been a major preoccupation of printers and that proofreaders are the genuine guarantors of the written language.

[Author's abstract]

Issue 32: "Journées GUTenberg 1999", Lyon

The issue includes papers presented at the recent GUTenberg annual meeting, which looked at both the specific and the general issues of TEX usage today. The specific subset of papers on 'TEX and XML' is reserved for *Cahiers* 33/34, a double issue. The papers outside that set are included here.

This issue was set in *ITC New Baskerville*, with *Gill Sans* for `\sf` and *Letter Gothic* for `\tt`. Of special interest are the sample pages of *SMF Baskerville*, a math font by Yannis Haralambous.

Thierry Bouche, Éditorial : TEX à l'approche du III^{e} millénaire : état des lieux et perspectives [Editorial: As we approach the third millennium ...]; pp. 3–4

The editor muses over the redirection of TEX's efforts from purely paper-based to the ever-expanding electronic permutations for displaying text and math. The articles in the issue are similarly quite broad in range, from beautiful typesetting (books, fonts, screen displays) to ever-improving tools (CDs and packages for French-language materials) to access TEX's capabilities, with a fair-sized detour to the world of musical notation.

As for the more specific focus of TEX and XML, the theme of GUTenberg's annual meeting, readers will find the conference papers in the next *Cahiers*, a double issue (no. 33/34).

Yannis Haralambous, Une police mathématique pour la Société mathématique de France : le *SMF Baskerville* [A math font for the French Math Society: SMF Baskerville]; pp. 5–20

The author describes in detail the evolution and design issues involved with creating a math Baskerville to work with the well-known text Baskerville. The introduction moves quickly but surely over what is becoming well-known ground, in terms of what is currently available as fully developed math fonts and current strategies to expand the repertoire of workable and aesthetically acceptable combinations of math and text fonts. The paper then moves through a brief history of the Baskerville font, and provides information on where the various commercial components can be acquired — this is not free-ware! And finally, the details dear to a font designer's heart, including a set of figures to compare a half-page of mathematics published by the Presses Universitaires de France with the same material set in SMF Baskerville, and closing with over 5 pages of Baskerville math, using examples from `testmath.tex`, an AMS test file.

Note: A reminder that another approach, that of combining elements from various fonts to arrive at a workable math font, was described in a recent issue of *TUGboat*. See Thierry Bouche, "Diversity in Math Fonts," *TUGboat* 19,2 (1998), pages 121–135. As well, in the same issue, on pages 176–187, Alan Hoenig described "Alternatives to Computer Modern Mathematics". The number of viable alternatives to the very complete CMR math fonts is rapidly expanding and everyone who works in mathematics typesetting should be heartened by all this activity.

Hàn Thế Thành, Améliorer la typographie de TEX [Improving TEX's typography]; pp. 21–28

This paper describes an attempt to improve TEX's typeset layout in pdfTEX, based on the adjustment of interword spacing after the paragraphs have been broken into lines. Instead of changing only the interword spacing in order to justify text lines, we also slightly expand the fonts on the line as well in order to minimise excessive stretching of the interword spaces. This font expansion is implemented using horizontal scaling in PDF. When such expansion is used conservatively, and by employing appropriate settings for TEX's line-breaking and spacing parameters, this method can improve the appearance of TEX's typeset layout.

[Author's abstract]

This is a translation (by Thierry Bouche) of the original paper, first presented at TUG'98 in Torún, Poland (August 1998). The article appeared in *TUGboat* 19,3 (1998), pages 284–288, where it was called "Improving TEX's Typeset Layout".

Laurent Guillopé, Statique et dynamique de documents mathématiques [Static and dynamic aspects of mathematics documents]; pp. 29–34

Various prototypes intended to examine a set of mathematics criteria are described. Even if none meet the contradictory requirements for this sort of numerical display, definite progress can, nevertheless, be noted.

Keywords: databank, reader, formula, mathematics, PDF, HTML, Internet

[Translation of French résumé]

A translation of the final paragraph of the introduction might help clarify things a little:

> This article is placed in the midst of the general framework of opposition between the static (books) and the dynamic (electronic representations). TEX, as *lingua franca* in the mathematical research community (amongst others) plays a pivotal role. And yet, it rapidly falls away in the face of the contradictory constraints of such displays; it is the resolution of these contradications which interests us here. Initial choices, preferred constraints, these yield quite different results.

José Grimm, Le rapport d'activité de l'Inria [Inria activity reports]; pp. 35–45

This article focusses on production of Inria's activity reports, starting with a collection of some 80 different LATEX documents, and then printed in 9 hardcopy volumes (totally some 2,294 A4-sized pages), translated into HTML via `latex2html` (3,131 web pages). Three document classes are used, along with three bibliography styles and two perl scripts. [Translation of author's résumé]

Of interest to anyone involved in very large-scale document production from multiple sources, and destined for multiple displays.

Fabrice Popineau, fpTEX : teTEX pour Win32 [fpTEX: teTEX for Win32]; pp. 47–61

The article provides an extensive overview of this port of teTEX for Windows machines, providing TEX users — and more particularly, TEX installers — with details on choices and decisions made regarding the development of the fpTEX distribution.

This article is a precursor to the paper which will be presented at TUG99, entitled "fpTEX: A teTEX-based distribution for Windows".

Daniel Flipo, Francisation d'un format LATEX : nouveautés [Adapting a LATEX format for French: updates]; pp. 63–70

TEX distributions based on Web2C v.7.x (teTEX for UNIX, fpTEX for Windows, CMacTEX for Mac), in conjunction with the revised `mltex.sty` package by Bernd Raichle, have considerably simplified the development and use of LATEX formats adapted for French-language applications. This report aims to examine some of the new possibilities. [Translation of author's résumé]

René Bastian, Figurations et notations de l'objet musical [Musical representation and notation]; pp. 71–90

Instead of giving a 'History of solutions' that composers have chosen to use, regarding musical notation, we will begin by highlighting a few extreme modes of notiation and then examine how some solutions, which appeared reasonable at the time, never got off the ground. This will be followed by a proposal for a grammar of musical exchange, one which might serve as a link between contemporary concerns and the traditional stock of notational symbols. [Translation of author's résumé]

— — * — —

Articles from *Cahiers* issues can be found in PostScript format at the following site (*note the new address*):

`http://www.gutenberg.eu.org/pub/gut/publications/publis.html`

About the *Cahiers*

Font use over the years. The GUTenberg group are currently producing issue 33/34. I've been doing these summaries since issue 12 (!). And in all this time, I have been quite blind to the fact that CMR is not the default font of choice. So, just for the record, here's a list of the main text fonts used in previous issues of the *Cahiers* — just one more reason to find a few copies (now available as downloadable `.pdf` files from their website) and see what a lot of *Apolline* or *Stone* looks like!

Most issues carry an explicit Colophon but I'd like to thank Jacques André for filling the gaps in this listing.

32	ITC New Baskerville, with Gill Sans for the `\sf` and Letter Gothic for `\tt`; several pages of SMF Baskerville
31	Apolline
30	Stone
28/29	CMR
27	Adobe Palatino
26	Adobe Palatino
25	Adobe Minion Multi Master
24	Adobe Palatino
23	CMR
22	Adobe Garamond
21	CMR
20	Mainly Univers, with some parts of Lucida and Omega
19	CMR
1–18	16 of these in Times

Change in website address. Another bit of news: the GUTenberg website has changed (noted in issue no. 31): `www.gutenberg.eu.org/pub`. Jacques tells me that GUTenberg intends to have all its publications on the website; to date, all of the *Lettres* are there, and the *Cahiers* start with issue no. 14.

[Compiled by Christina Thiele]

Calendar

1999

Aug 8 – 13	SIGGRAPH 99, Los Angeles, California. For information, visit `http://www.siggraph.org/s99/`.
Aug 15 – 19	**TUG'99** — The 20th annual meeting of the TeX Users Group, "TeX Online — Untangling the Web and TeX", University of British Columbia, Vancouver, Canada. The Web page, `http://www.tug.org/tug99/`, is updated regularly.
Aug 23	*TUGboat* **20** (3), deadline for reports and news items.
Aug 30 – Sep 2	15th International Unicode Conference, San Jose, California. For information, visit `http://www.unicode.org/unicode/iuc15/`.
Aug 30 – Sep 3	Seybold San Francisco/Publishing 99, San Francisco, California. For information, visit `http://www.seyboldseminars.com/Events`.
Sep 4 – Oct 17	ABeCeDarium: A traveling juried exhibition of contemporary artists' alphabet books by members of the Guild of Book Workers, appearing at the Vida Ellison Gallery, Denver Public Library, Denver, Colorado. Sites and dates are listed at `http://palimpsest.stanford.edu/byorg/gbw`.
Sep 13 – 14	EGUTH'99: First meeting of the Spanish-speaking TeX Users Group (CervanTeX), Universidad Politécnica de Madrid, Spain. For information, visit `http://feynman.faii.etsii.upm.es/~eguth99`.
Sep 12 – 13	UK-TUG Autumn meeting and 10th AGM: TeX/LaTeX and their relationship to SGML/HTML/XML, London, UK. For information, contact `uktug@mail.rhbnc.ac.uk`.
Sep 19	DANTE, 21st meeting, Heidelberg University, Germany. For information, contact `dante@dante.de`.
Sep 20 – 23	EuroTeX '99, the XIth European TeX Conference, "Paperless TeX", Ruprecht-Karls University, Heidelberg, Germany. Tutorials will precede and follow the main conference. For information, visit `http://uk.tug.org/EuroTeX-99/`.
Sep 23 – 24	H2PTM99, the 5th Conference on Hypertexts and Hypermedia: Products, Tools, Methods, Saint Denis, Paris, France. For information, visit `http://www.labart.univ-paris8.fr/~conf99/`.
Oct 7 – 10	ATypI '99, Association Typographique Internationale, Boston, Massachusetts. For information, visit `http://www.atypi.org/`.
Oct 22 – 24	The 24th Annual Conference of the American Printing History Association, "A Century of Book Design in Europe and America: Printing, Practitioners, and Presses", The Grolier Club, 47 East 60th Street, New York, New York. For more information, visit `http://wally.rit.edu/cary/apha.html`.
Nov 3 – Dec 17	ABeCeDarium: A traveling juried exhibition of contemporary artists' alphabet books by members of the Guild of Book Workers, appearing at the Denison Library, Scripps College, Claremont, California. Sites and dates are listed at `http://palimpsest.stanford.edu/byorg/gbw`.
Nov 8	*TUGboat* **20** (4), deadline for technical submissions.
Nov 22	*TUGboat* **20** (4), deadline for reports and news items.
Dec 6 – 9	XML 99, Philadelphia, Pennsylvania. For information, visit `http://www.gca.org/conf/conf1996.htm`.

Status as of 30 June 1999

For additional information on TUG-sponsored events listed above, contact the TUG office (+1 503 223-9994, fax: +1 503 223-3960, e-mail: `office@tug.org`). For events sponsored by other organizations, please use the contact address provided.

Additional type-related events and news items are listed in the Sans Serif Web pages, at `http://www.quixote.com/serif/sans`.

2000

Feb 7	*TUGboat* **21** (1), deadline for technical submissions.
Feb 7 – 11	Seybold Seminars Boston/ Publishing 2000, Boston, Massachusetts. For information, visit `http://www.seyboldseminars.com/Events`.
Feb 21	*TUGboat* **21** (1), deadline for reports and news items.
Mar 8 – 10	DANTE 2000 and 22nd meeting, Technische Universität Clausthal-Zellerfeld, Germany. For information, contact `dante2000@dante.de`.
Apr 11	*TUGboat* **21** (2), deadline for technical submissions.
May 9	*TUGboat* **21** (2), deadline for reports and news items.
Jun 16 – 18	TypeCon 2000, Westborough, Massachusetts. For information, visit `http://tjup.truman.edu/sota/`.
Jun 22 – 24	TypoMedia 2000, “Future of Communication”, Mainz, Germany. Linotype’s design conference; for information, visit `http://www.typomedia.com`.
Jul 23 – 28	SIGGRAPH 2000, New Orleans, Louisiana. For information, visit `http://www.siggraph.org/calendar/`.
Aug 12 – 18	**TUG 2000** — The 21st annual meeting of the TEX Users Group, “TEX enters a new millennium”, Wadham College, Oxford, UK. For information, visit `http://tug2000.tug.org/`.
Aug 28 – Sep 1	Seybold San Francisco/ Publishing 2000, San Francisco, California. For information, visit `http://www.seyboldseminars.com/Events`.
Sep 12	*TUGboat* **21** (3), deadline for reports and news items.
Sep 13 – 15	DDEP: Digital Documents and Electronic Publishing (successor to EP98) and WEPT: Week on Electronic Publishing and Typography, Munich, Germany. For information, visit `http://www.irisa.fr/ep98`.
Sep 19	*TUGboat* **21** (4), deadline for technical submissions.
Oct 17	*TUGboat* **21** (4), deadline for reports and news items.

Late-Breaking News

Production Notes

Mimi Burbank

”We’re late! We’re late!” . . .[1]

One of the more daunting aspects of producing a journal is getting all of the material together in a timely fashion. My job literally ends up being “mushing files together to get a contiguous set of pages which add up to some multiple of 8 or 16.” For this issue, promised material did not appear in a timely fashion, there were various problems with reviewers and then everyone began getting ready to go to TUG ’99. I seem to be just “full” of excuses . . .

Any issue that deals with fonts, and includes multiple graphics always presents some kind of challenge in terms of production. Often it involves a comedy of errors but I always consider the actual production of *TUGboat* to be my “Continuing Education”.

The article by Bogusław Jackowski (page 104) required additional fonts, and METAFONT sources were provided by the author.

Output. The final camera copy was prepared at SCRI using the *TEX Live* 4 setup, which is based on the *Web2c* TEX implementation version 7.3 by Karl Berry and Olaf Weber. PostScript output, using outline fonts, was produced using Radical Eye Software’s dvips(k) 5.85, and printed on an HP LaserJet 4000 TN printer at 1200dpi.

Coming In Future Issues The next issue will contain the proceedings of the TUG ’99 Annual Meeting, held in Vancouver, BC.

◇ Mimi Burbank
SCRI, Florida State University,
Tallahassee, FL 32306 – 4130
mimi@scri.fsu.edu

[1] The image comes from http://www.disneyclipart.com/Movies/Alice_in_Wonderland/White_Rabbit/.

Institutional Members

American Mathematical Society, *Providence*, *Rhode Island*

CNRS - IDRIS, *Orsay*, *France*

College of William & Mary, Department of Computer Science, *Williamsburg*, *Virginia*

CSTUG, *Praha*, *Czech Republic*

Florida State University, Supercomputer Computations Research, *Tallahassee*, *Florida*

Hong Kong University of Science and Technology, Department of Computer Science, *Hong Kong*, *China*

IBM Corporation, T J Watson Research Center, *Yorktown*, *New York*

ICC Corporation, *Portland*, *Oregon*

Institute for Advanced Study, *Princeton*, *New Jersey*

Institute for Defense Analyses, Center for Communications Research, *Princeton*, *New Jersey*

Iowa State University, Computation Center, *Ames*, *Iowa*

Kluwer Academic Publishers, *Dordrecht*, *The Netherlands*

KTH Royal Institute of Technology, *Stockholm*, *Sweden*

Los Alamos National Laboratory, University of California, *Los Alamos*, *New Mexico*

Marquette University, Department of Mathematics, Statistics and Computer Science, *Milwaukee*, *Wisconsin*

Masaryk University, Faculty of Informatics, *Brno*, *Czechoslovakia*

Max Planck Institut für Mathematik, *Bonn*, *Germany*

New York University, Academic Computing Facility, *New York*, *New York*

Princeton University, Department of Mathematics, *Princeton*, *New Jersey*

Space Telescope Science Institute, *Baltimore*, *Maryland*

Springer-Verlag Heidelberg,, *Heidelberg*, *Germany*

Stanford University, Computer Science Department, *Stanford*, *California*

Stockholm University, Department of Mathematics, *Stockholm*, *Sweden*

University of Canterbury, Computer Services Centre, *Christchurch*, *New Zealand*

University College, Cork, Computer Centre, *Cork*, *Ireland*

University of Delaware, Computing and Network Services, *Newark*, *Delaware*

Universität Koblenz–Landau, Fachbereich Informatik, *Koblenz*, *Germany*

University of Oslo, Institute of Informatics, *Blindern*, *Oslo*, *Norway*

University of Texas at Austin, *Austin*, *Texas*

Università degli Studi di Trieste, *Trieste*, *Italy*

Uppsala University, Computing Science Department, *Uppsala*, *Sweden*

Vanderbilt University, *Nashville*, *Tennessee*

Vrije Universiteit, *Amsterdam*, *The Netherlands*

TEX Consulting & Production Services

Information about these services can be obtained from:

TEX Users Group
1466 NW Naito Parkway, Suite 3141
Portland, OR 97209-2820, U.S.A.
Phone: +1 503 223-9994

Fax: **+1 503 223-3960**
Email: office@tug.org
URL: http://www.tug.org/consultants.html

North America

Hargreaves, Kathryn
135 Center Hill Road,
Plymouth, MA 02360-1364;
(508) 224-2367; letters@cs.umb.edu
I write in TEX, LATEX, METAFONT, MetaPost, PostScript, HTML, Perl, Awk, C, C++, Visual C++, Java, JavaScript, and do CGI scripting. I take special care with mathematics. I also copyedit, proofread, write documentation, do spiral binding, scan images, program, hack fonts, and design letterforms, ads, newsletters, journals, proceedings and books. I'm a journeyman typographer and began typesetting and designing in 1979. I coauthored *TEX for the Impatient* (Addison-Wesley, 1990) and some psychophysics research papers. I have an MFA in Painting/Sculpture/Graphic Arts and an MSc in Computer Science. Among numerous other things, I'm currently doing some digital type and human vision research, and am a webmaster at the Department of Engineering and Applied Sciences, Harvard University. For more information, see: http://www.cs.umb.edu/ kathryn.

Loew, Elizabeth
President, TEXniques, Inc.,
675 Massachusetts Avenue, 6th Floor,
Cambridge, MA 02139;
(617) 876-2333; Fax: (781) 344-8158
Email: loew@texniques.com
Complete book and journal production in the areas of mathematics, physics, engineering, and biology. Services include copyediting, layout, art sizing, preparation of electronic figures; we keyboard from raw manuscript or tweak TEX files.

Ogawa, Arthur
40453 Cherokee Oaks Drive,
Three Rivers, CA 93271-9743;
(209) 561-4585
Email: Ogawa@teleport.com
Bookbuilding services, including design, copyedit, art, and composition; color is my speciality. Custom TEX macros and LATEX2$_\varepsilon$ document classes and packages. Instruction, support, and consultation for workgroups and authors. Application development in LATEX, TEX, SGML, PostScript, Java, and ßC++. Database and corporate publishing. Extensive references.

Outside North America

DocuTEXing: TEX Typesetting Facility
43 Ibn Kotaiba Street,
Nasr City, Cairo 11471, Egypt
+20 2 4034178; Fax: +20 2 4034178
Email: main-office@DocuTeXing.com
DocuTEXing provides high-quality TEX and LATEX typesetting services to authors, editors, and publishers. Our services extend from simple typesetting and technical illustrations to full production of electronic journals. For more information, samples, and references, please visit our web site: http://www.DocuTeXing.com or contact us by e-mail.