TUGboat

Volume 23, Number 1 / 2002
2002 Annual Meeting Proceedings

TEX Users Group

Memberships and Subscriptions

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

2002 dues for individual members are as follows:

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

Membership in the TEX Users Group is for the calendar year, and includes all issues of *TUGboat* for the year in which membership begins or is renewed. Individual membership is open only to named individuals, and carries with it such rights and responsibilities as voting in TUG elections. For membership information, visit the TUG web site: `http://www.tug.org`.

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

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

Institutional Membership

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

TEX is a trademark of the American Mathematical Society.

Printed in U.S.A.

Addresses

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

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

Telephone

+1 503 223-9994

Fax

+1 503 223-3960

Electronic Mail (Internet)

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

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

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

To contact the Board of Directors:
`board@tug.org`

World Wide Web

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

Problems not resolved?

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

[printing date: March 2003]

2002 Annual Meeting Proceedings

TEX Users Group
Twenty-third Annual Meeting
Trivandrum, Kerala, India
September 4-7, 2002

TUGBOAT

COMMUNICATIONS OF THE TEX USERS GROUP

TUGboat Editor Barbara Beeton
Proceedings Editors Kaja Christiansen
Karl Berry

Volume 23, Number 1 • 2002
Portland • Oregon • U.S.A.

Welcome to the TUG 2002 Proceedings!

In early September 2002, at the southwest tip of the Indian subcontinent overlooking the Arabian Sea, a multinational gathering of delegates[1] met at the 23rd annual conference of the TEX Users Group. We were in Thiruvananthapuram (or Trivandrum), the capital of the stunning state of Kerala. Many had arrived already on September 1st, to participate in excellent tutorials and workshops by C.V. Radhakrishnan (India), Hans Hagen (Netherlands) and Lou Burnard and Sebastian Rahtz (Oxford).

On the official opening day of September 4th the delegates were greeted by a small, but very real, elephant, and traditional music. The conference was launched and the work could begin. A great success! Inspiring presentations and flawless organisation, combined with the kindness of our Indian friends and the beauty of the surroundings made it an unforgettable experience.

For an overview of the meeting and the presentations in particular, see:

- *Report of the 23rd Annual Meeting and Conference* at `http://www.tug.org.in/tug2002`
- *Report and Travelogue* by Ross Moore (in this issue).

Those who could not attend TUG 2002 will, I hope, find this issue interesting and very much worth reading. Those of you who were there will enjoy the walk through the proceedings and look back at TUG 2002 as an inspiring and rewarding conference.

Producing the proceedings

In the midsummer of 2002, Sebastian Rahtz asked whether I would like to work on the conference proceedings; I gladly accepted, seeing it as a privilege and a challenge. The actual work started shortly after that: many evenings of this Danish summer were spent typing away...

We received 25 submissions. One in plain TEX, one to be run with Omega and the rest in LATEX. The articles were processed with the post-TEX Live 7 setup. In August 2002, and with the printing house deadline over my head, I transferred the camera-ready PDF files for preprints to C.V. Radhakrishnan at Focal Image (India).

The work on conference proceedings started in the late fall of 2002 and continued during the winter holidays. A few of the papers were resubmitted (the last one as late as November); all needed to be edited/reviewed and there was an ongoing correspondence between the authors and Karl Berry and me on the suggested changes.

In January 2003, the TEX Live 7 setup, tailored for the 2002 proceedings, was made at CSIT at Florida State University where all papers were uploaded for the final production of this issue.

TUG 2002 on the Web

After the printed copy of *TUGboat* is distributed, the conference proceedings will be available online in the *TUGboat* area of the TUG web site:

`http://tug.org/tugboat`

Also, make sure to visit the TUGIndia site at:

`http://www.tug.in.org/tug2002`

for the post-conference bulletin with report, slides, newspaper clippings and photo gallery.

Acknowledgments

Many thanks to Sebastian Rahtz for tirelessly answering my questions; to Barbara Beeton, Mimi Burbank, Robin Fairbairns, Christina Thiele and Dominik Wujastyk for reviewing the papers; and to Karl Berry, for his support and encouragement.

Finally, I give my thanks to the Computer Science Department and the BRICS Research Centre, University of Aarhus, for making my attendance at TUG 2002 possible. This was a unique experience, both professionally and personally.

◇ Kaja Christiansen

– – ∗ – –

Note from the Editor

This issue of *TUGboat*, containing the proceedings of TUG 2002, will reach you before the last issue of the 2001 volume. There are a couple of reasons for this: *TUGboat* in general has suffered delays, as mentioned in the "Editorial Comments" column of Vol. 22, Nos. 1/2, and the editorial group working on the TUG 2002 proceedings has been particularly effective in carrying out the tasks associated with the issue's production.

For these reasons, we are releasing the completed proceedings issue as soon as it is ready, rather than delaying it artificially to follow the nominal sequence.

The next issue to appear *will* be Vol. 22, No. 4 (December 2001); production is underway in parallel with this issue. We ask for your understanding in this matter.

◇ Barbara Beeton

[1] Attendees came from Australia, Canada, Czech Republic, China, Denmark, France, Germany, India, Iran, Netherlands, Poland, Switzerland, UK and USA.

TUG 2002, Thiruvananthapuram Report and Travelogue

Ross Moore

While wandering among the shops at Singapore's extensive Changi airport, after an 8-hour flight from Sydney, it was not such a surprise to hear my name being called to come to the Info desk. That would be Wendy McKay, whom I'd agreed to meet there. We were both taking the same flight directly to Thiruvananthapuram, abode of the sacred snake Anantha. This city, also known as Trivandrum, is the capital of Kerala State in South India, where TUG 2002, the 23rd annual TEX Users Group conference, was to be held. Wendy had come further than I, from Los Angeles via Taiwan. Having more hours to kill, she had taken a couple of sight-seeing tours around Singapore. With just three flights per week, Silk Air provides a convenient way to get to Trivandrum, when coming from further East, without the need to pass through other cities or rely on transport connections within India itself. Other conference attendees would be traveling even further to get to Kerala. In all there were 30 'foreigners' coming from 14 different countries. For most it would be their first trip to India. With 20–30 from elsewhere in India, local attendees at the workshops and the organisers themselves, the total number of participants would be close to 80.

Upon exiting the airport at Thiruvananthapuram it was a pleasure to be greeted by Kaveh Bazargan and C. V. Radhakrishnan. They run Focal Image India and provided the inspiration for holding TUG 2002 in Trivandrum, and the driving force behind its organisation. Hotel Samudra, where most of the foreign delegates would be staying, is a half-hour drive away at Kovalam. It was Radhakrishnan's car, but Kavah was to drive. This may not seem remarkable unless you have already experienced driving in India. Roads are generally quite narrow, with barely enough room to pass another vehicle. Traffic drives on the left but the central line, if marked at all, serves only as an indicator. During the day roads are very busy, being used by people and bicycles — sometimes laden high with boxes, milk crates or caged chickens — as well as motorised vehicles. The latter includes motor scooters, motor bikes — lady passengers sit side-saddle, perhaps carrying a baby; whole families of 4 or 5 somehow find a place to sit — and the small 3-wheeled auto-rickshaws, as well as cars, buses and trucks. Occasionally there is a cart: two wheelers pulled by a person, or the larger 4-wheeled variety powered by a pair of bullocks. Elephants using the road are rare within towns and cities, but can be encountered working along the highways between towns.

Vehicles pass with only inches to spare; external rear-vision mirrors are not used, as they would quickly be smashed against another vehicle. When desiring to pass, the rule is to honk the horn to let the driver in front know of your presence. Smaller vehicles give way to larger vehicles, more as a matter of self-preservation than as a rule. A common way to turn right is to cross and drive the last 50 metres or so down the opposite side of the road, before making the turn. There can be vehicles traveling in the opposite direction on *both* sides of your car, so it's no wonder that mirrors are of little use— there's no time to look into them. The traffic signs say 'Live and let live' and list five different speed limits for the different types of motorized vehicle. Normally foreigners do not drive, but instead hire a taxi with a local driver. A whole afternoon of sightseeing can cost less than $10. The driver will wait while you visit museums, temple, art galleries, the zoo, do some shopping or stop at a roadside coconut stand. Kaveh has spent so much time in India now that he now dares to sit behind the wheel; when I visited him last year he would always use a driver.

Hotel Samudra, at Kovalam, is run by the Kerala Tourist Development Commission (KTDC), a branch of the State Government. It sits atop a small hill with spectacular views across the ocean on one side. Apartments run down the hill at the back, with a short path leading down among the coconut palms to a small sandy beach. Paths lead to other beaches where every morning the fishermen come in with their small wooden boats, laden with fresh fish, after spending the night on the sea with their nets. There are plenty of hotels, but June to September is off-season for the tourist trade. During the week of TUG 2002, conference delegates were

practically the only occupants, apart from the teams of workers that ran the hotel and restaurant, the Ayurvedic massage center, and taxi drivers waiting to offer their service. A special rate of roughly $20 per night had been negotiated for the duration of the conference.

'Stand up and be Proud of TEX' was the motto for the TUG 2002 conference, organised by the Indian TEX Users Group, Free Software Foundation — Indian Chapter and the Department of Information Technology, Government of Kerala. They have every reason to be proud of their efforts. This was certainly the best run TEX Users Group meeting that I have attended. Many other delegates agree. Site for the talks was the Technopark in Koryavattom, a little to the north of the city of Trivandrum. As Kovalam being 17kms to the south, this meant a 40 minute bus trip each morning, so that each day started with the excitement of road traffic in India. Delegates from elsewhere on the sub-continent mostly were staying at a hotel in Trivandrum itself; they arrived on a separate bus or by car. The main building, with auditorium and conference rooms, has a red-tiled roof and is part of a cluster on top of a large hill with views to the north across a sea of coconut palms and cashew-nut trees. Huts and houses are invisible beneath the green canopy.

On the first day delegates were delighted to be greeted by a traditional band of blaring horns and beating drums and an elephant decked in ceremonial dress with a silk banner printed with the conference logo. Similar banners hung from light poles at the entrance to Technopark, and along the driveways. Fresh coconuts were available inside. Kaveh was dressed in a dhoti. Igor Rodionov was first to ride the elephant, followed by Gyöngyi Bujdosó, before it was time for the opening session. Satish Babu (of Internet Applications Technologies Pvt. Ltd., Technopark), as head of the Organising Committee, chaired the opening session. Short addresses were given by Ajay Shah (Ministry of Finance, Government of India, New Delhi) and Dr K. R. Srivathsan (International Institute for Information Technology and Management, IIIT), Technopark, Trivandrum), representing the major sponspors for TUG 2002. These were followed by the keynote address by Ajit Ranade (Chief Economist, ABN AMRO Bank, Mumbai) giving an overview of the 'Status of TEX in India'. Ajit first used TEX as a graduate student and presented names of many publishers which use TEX within their production processes in India, mostly without explicit recognition of this aspect of their work. S. Rajkumar (Lanscape Technologies, Trivandrum) followed by giving a description of the challenges and opportunities for TEX with regard to Indic typesetting, with its many languages and scripts. There are over 5000 commercial fonts in daily use, but only 20 or so with support currently available in TEX. Next Amitabh Trehan described his experiences with TEX, typesetting in Hindi, Sanskrit and Persian. This included a project preparing a book of Indian verse, with commentary, prepared entirely in LATEX. This session finished with Gyöngi describing special aspects of typesetting in Hungarian, and the difficulties faced in trying to adapt LATEX to account for these.

The auditorium at Technopark is shaped like an amphi-theatre with a large stage occupying one corner. Delegates are seated at long desks arranged as quarter-circles facing the stage, rising and of increasing length towards the back of the room. Entrance is from one side, at the top. Microphones on the desks are available for each pair of seats. A data-projector was mounted in the central aisle, projecting onto a large screen on-stage. Most speakers used this, and all talks were recorded for both audio and video. On either side of the screen hung silk banners proclaiming the conference, organisers and sponsors. Another smaller banner, with the conference logo, hung from the podium. Internet access was available to delegates via the IIIT, located downstairs in the same building but requiring a short walk out the front, and down a driveway to the separate entrance.

Over the three days prior to the conference proper there had been four workshops. The first of these was given by C. V. Radhakrishnan on Sunday, at the offices of Focal Image India. This was a hands-on introduction to TEX and LATEX, in training sessions over the entire day, with participants from around India. Each received a copy of the book 'LATEX Tutorials: A Primer to LATEX 2_ε', prepared by the Tutorial Team of the Indian TEX Users Group, and printed locally. These tutorials can also be found on the web[1]. A revised version (with corrections and extra chapters) will shortly be published by the Free Software Foundation[2], to become their official manual for LATEX.

On Monday morning, C. V. Radhakrishnan held a workshop on 'LATEX to SGML/XML conversion'; describing the theory behind techniques that are put into practice in the day-to-day work at Focal Image India. This and subsequent workshops were held in a meeting room adjacent to the auditorium at Technopark. Lunch was provided, buffet-style, with several mildly spicy local dishes of fish,

[1] http://www.tug.org.in/tutorials.html
[2] http://www.fsf.org/

chicken and exotic vegetables and breads. Delicious desserts with ice-cream cooled the palate. Soups and some Western dishes were also available each day. That afternoon Hans Hagen (Pragma ADE, The Netherlands) demonstrated METAPOST and pdfTEX, showing how easy it is to create graphics, both easy and some quite complicated ones. for use within documents prepared using ConTEXt.

Tuesday morning's newspaper contained a report of the Press Conference held the night before. TUG 2002 was in the news: 'Free Software typesetting comes to Kerala', it proclaimed. Just a single tutorial occupied the whole day. Sebastian Rahtz and Lou Bernard (Oxford University Computing Services) took it in turns to explain the ins-and-outs of the Text Encoding Initiative (TEI). This is a method to generate DTDs for XML documents in any particular field. It can help enormously in the need to be able to archive, and subsequently access to recover in printed or on-screen form, academic documents and technical manuals. Indeed, any type of 'document' should be able to be handled using these methods.

In the evenings, dinner at Hotel Samudra was served buffet-style, outside on a lawn seated under coconut trees with a view westwards, across the ocean and the setting sun. Different dishes were prepared each night, so that after a week all the dishes available from the restaurant must have been sampled. When ordering a beer, asked for 'chilled' beer, else it will not be cold. Tuesday night was registration; each participant received a coir satchel and folder, manufactured from the hard stringy fibres of coconut shells. On these was printed the conference logo, and they came stuffed with promotional materials about Kerala, a nicely bound preprint book, and two CDs of TEX-related and other Free Software, including the very latest version of TEXLive 7, and tools for using the TEI and XML. Traditional Indian entertainment was provided on two nights. This included spectacular fire-throwing, Indian martial arts, dancing girls in bright costumes, and flute Ragas. Dominic Wujastyk (Wellcome Centre, University College London) used his extensive knowledge of Indian customs and culture to provide information on what we were about to see, explaining the traditional stories being depicted. The large swimming pool behind the main hotel buildings received much use, both at night and during the day.

Throughout the conference, morning and afternoon teas and lunches were provided at Technopark. One could stand, chat and eat in the large room used as the serving area, or go outside onto the large balcony where chairs were placed under huge marquées. This balcony afforded also a spectacular view across the coconut and cashew trees; frequently hawks could be seen hovering, sometimes coming close to the balcony. In the afternoon of the first day of the conference, Satish Babu (Computer Society of India) gave an Indian perspective of 'New Horizons of Free Software'. This is very important to the State of Kerala, where the literacy rate is very high but incomes are low. India as a whole is rich in programmer resources, but is otherwise poor; the Free Software model is appropriate for the needs of governments, institutions and industry. Wlodzimierz Bzyl (University of Gdansk, Poland), speaking on 'The Tao of Fonts', showed some beautiful pictures of letter-like shapes used in different cultures. The talks for the day finished with Roozbeh Pournader (Sharif University of Technology, Tehran, Iran) presenting a general outline of The Unicode Standard, emphasizing recently introduced features. The important message here is to be Compliant; an aspect that the Linux and gnu communities embrace, and now backed also by corporations such as SUN and IBM, but where Microsoft has got some of the ideas wrong in existing software.

The burning issue in the newspapers that week, and the previous week-end, had been a hike in the cost of power, electricity and gas. The reason given was reduced availability of water for hydro-electricity as monsoonal rains had been weak this season. These price-hikes had caused massive protests throughout the state, including the ransacking of government offices and damage to more than 100 buses. We had seen none of this, but the issue was to affect the conference, as a general strike had been called for the Friday. It would be unsafe to be on the roads, particularly as the bus trip from Kovalam to Technopark passed through at least two major rallying points. Besides, the drivers would not be working. A revised conference schedule was prepared, in which Friday would now be a free day. Some talks were withdrawn or moved to Saturday, and there would be an effort made to organise extra workshops and discussion groups at Hotel Samudra for Friday.

Thursday morning was 'Mathematics Morning'. This started with a keynote talk by Hans Hagen, titled 'ConTEXt, XML and TEX: State of the Art?'. What has this to do with mathematics? He discussed the publication process, and one of his main examples was a journal for the Dutch Mathematical Society, having a rather flexible format allowing figures and equations to be displayed spanning either one, two or all three of the columns in a wide page layout. Next David Kastrup (Bochum, Germany)

compared five different TEX editing systems which have varying degrees of WYSYWG-like features for displaying typeset mathematics within the editor window. In my own talk I showed a way to embed the source code for LATEX mathematics within a PDF document, using text-fields that can be shown and hidden using roll-over actions in the Acrobat Reader window. A search-engine was implemented, also in JavaScript, to allow short strings of mathematics source to be located within the text-fields. How far to expand user-defined macros emerges as a conceptual issue with this approach. The morning finished with Stephen Watt (University of Western Ontario, London, Canada) talking about issues in the conversion between TEX and MathML, and the need to conserve implicit mathematical semantics. Again the issue of not expanding macros is a key aspect.

After lunch Karel Píška displayed the Type 1 versions of Indic fonts that he had created using TEXtrace, applied to fonts available on CTAN as METAFONT versions only. With some extra editing to simplify the outline paths and remove unwanted artifacts, he has produced 55 scalable fonts suitable for inclusion in PDF documents. Next Behdad Esfahbod (Sharif University of Technology, Tehran, Iran) spoke about FarsiTEX and the Iranian TEX community. A new release will include Type 1 fonts for Persian and Arabic scripts, though proper support in editors for bi-directional typesetting remains the biggest problem. Denis Roegel (LORIA, Nancy, France) showed how to use METAOBJ, a high-level object-oriented graphics language implemented with METAPOST. He discussed the advantages of an object-oriented approach to graphics, as well as describing the new primitives defined in METAOBJ. The day finished with Karel Skoupy (ETH, Zurich, Switzerland) outlining his vision of what would be required in a new typesetting system, using computing techniques developed and recognised as being valuable, since the advent of TEX. This talk generated much lively discussion.

During the day it was learnt that the general strike for the next day had been called off. This was due to the government capitulating by removing the hike in energy prices, due to the unified opposition from across all the political parties. Whether the reported failure of the energy board to collect substantial amounts of revenue, from corporations and other government agencies, had any bearing on this reversed decision was not at all clear. In any case, the TUG 2002 organisers decided to stick with the revised schedule for Friday and Saturday. David Kastrup gave the tutorial that he had been unable to give earlier in the week. There were lively discussions about matters concerning TEX user-groups around the world, and many delegates took advantage of the Internet connection at Hotel Samudra, or spent much of the day shopping or enjoying the beach or swimming pool. Wendy and I visited Focal Image India to consult with C. V. Radhakrishnan, receiving a detailed demonstration of the methods he developed to convert LATEX source to XML, and back again to recover consistent high-quality printable output.

The TUG Business meeting, chaired by yours truly, was first on the agenda for Saturday morning. Here we skimmed through the draft minutes of the Board of Directors meeting, held in Portland, Oregon on 20th July. Some factual errors were noted and concern was expressed at the lateness of *TUGboat* issues, and the declining number of members of TEX Users Group, over the last couple of years. TEX Users Group board member Kaja Christiansen (University of Århus, Denmark) described the new TEX Development Fund. Members were advised of the forthcoming elections in 2003 and invited to consider making nominations for President and Board members. Following the break for morning tea were two talks coming from the IIIT group at Technopark; namely G Nagarjuna spoke on 'Symantic Web, GNOWSYS and Online Publishing' followed by Dr K. R. Srivathsan describing the 'Education Grid'. Then Hong Feng (President of CTUG, People's Republic of China) described efforts by CTUG to implement the Chinese artificial logical language Lojban, using TEX. A long morning finished with K. Anilkumar (Linuxense Information Systems, Trivandrum, India) describing how to use shell-escapes to make TEX read a database, and generate reports.

After lunch John Plaice (School of Computer Science and Engineering, UNSW, Sydney, Australia) departed from his advertised talk on Devanagari support in Omega, to speak on some completely different aspects of Omega development. Fabrice Popineau (SUPELEC, Metz, France) described the new aspects of TEXLive 7 for Windows' users, then Karel Skoupy demonstrated TEX running as a web service. After a break for afternoon tea, the strains of Hawaiian music wafted through the auditorium, before Wendy and I gave a presentation about next year's meeting, TUG 2003, to be held in Hawaii towards the end of July. This was a walk through the web pages, now on www.tug.org, and the screening of a tourist video of the sight and delights to be found on the large island. Dominic Wujastyk (Wellcome Centre, University College London) closed the

meeting with heartfelt thanks to the organisers for a most enjoyable and stimulating conference, well-run by the organisers and successful in every way. No major problems had been encountered; the potential for disruption due to the strike threat had been effectively accommodated with a minimum of fuss.

Throughout the conference there was a line of desks outside the auditorium, where tour-operators provided information and touted their business. Kerala is famous for its 'backwater cruises', on the large lake and waterways near Alappuzha (Alleppey) and Kochi (Cochin). This is the rice-growing area, roughly 100 miles north of Trivandrum. Tourists hire a boat — converted rice transports, with elaborate superstructure constructed from bamboo poles and weaving, propelled by a single out-board motor. There can be one, two or three bedrooms, for sleeping on-board over one or two nights, and a comfortable sitting-room/living area. The crew consists of 3 or 4, including a cook, with all meals provided. Some of the more modern boats have solar panels for the electric lighting. Several groups of TEX Users Group delegates took cruises starting Sunday afternoon, taking cruises that did a large loop across the lake and around paddy fields.

We met up on the water, and at landing points. Sights along the cruise included churches, temples, villages and small farming plots, as well as exotic flowers, plants and bird-life.

We were treated to a spectacular sunset. A crew of 65 were practising for the famous 'snake-boat' races; their boat was not a full-sized snake-boat, which has a crew of more than 100 oarsmen.

Included in some of the tours is a visit to a high-quality hotel, either for lunch or an overnight stay, or to other popular tourist destinations. In our case it was the Raja Garden Retreat at the beach resort town of Varkala. Along the way we stopped to photograph an elephant feeding on coconut palm fronds, while taking a break from moving huge logs just off to the side of the highway. Seeing us tourists, the workers asked for 100 Rupees (≈$2) to bring the elephant closer.

Back at Hotel Samudra, for one last night, there were still a few TEXies. Some others returned the next day from their tours. After a last visit to Focal Image India, and a bit more shopping, there was enough time for a rejuvenating Ayurvedic massage — very oily. There's always more to pack for the return journey; dinner of Kerala cuisine in the restaurant, then a taxi took us to the airport for the 11.00pm flight. TUG*India* had done a great job; thanks especially to Kaveh and C. V. Radhakrishnan. Hopefully the opportunity will arise to return to "God's Own Country", in the not too distant future.

Feeding the elephant

TUG 2002 Programme
Stand up and be proud of TEX!

Wednesday, September 4, 2002

Opening Ceremony

Ajit Ranade
Status of TEX in India – An Overview

S. Rajkumar
Indic Typesetting – Challenges and Opportunities

Wagish Shukla and Amitabh Trehan
Typesetting in Hindi, Sanskrit and Persian: A Beginner's Perspective

Gyöngyi Bujdosó and Ferenc Wettl
On the Localization of TEX in Hungary

Satish Babu
New Horizons of Free Software: An Indian Perspective

Włodzimierz Bzyl
The Tao of Fonts

Roozbeh Pournarder
Catching up to Unicode

Thursday, September 5, 2002

Hans Hagen
ConTEXt, XML and TEX: State of the Art?

David Kastrup
Revisiting WYSIWYG Paradigms for Authoring LATEX

Ross Moore
serendiPDF with Searchable Math-fields in PDF Documents

Stephen M. Watt
Conserving Implicit Mathematical Semantics in Conversion between TEX and MathML

Karel Píška
A Conversion of Public Indic Fonts from METAFONT into Type 1 Format with TEXTRACE

Behdad Esfahbod and Roozbeh Pournader
FarsiTEX and the Iranian TEX Community

Denis Roegel
METAOBJ: Very High-Level Objects in METAPOST

Karel Skoupy
New Typesetting Language and System Architecture

Friday, September 6, 2002

Workshop: Tutorial by *David Kastrup*

Saturday, September 7, 2002

TUG Business Meeting

G. Nagarjuna
Semantic Web, GNOWSYS and Online Publishing

K. R. Srivatsan
Education Grid

Hong Feng
The Marriage of TEX and Lojban

K. Anil Kumar
TEX and Databases – TEXDBI

Yannis Haralambous and John Plaice
Low-level Devanāgarī Support for Omega — Adapting `devnag`

Fabrice Popineau
TEX Live under Windows: what's new with the 7th edition?

Karel Skoupy
TEX File Server

Closing Ceremony

Conference Auditorium

Participants at the 23rd Annual TUG Meeting
September 4–7, 2002, Trivandrum, Kerala, India

Anil Kumar, K
Linuxense Information Systems
Trivandrum, India
Anil@linuxense.com

Bazargan, Kaveh
Focal Image Ltd.
Exeter, UK
kaveh@focalimage.com

Brahmanayagam, K
Linuxense Information Systems
Trivandrum, India
Kb@linuxense.com

Bujdosó, Gyöngyi
Institute of Mathematics
and Informatics
University of Debrecen, Hungary
ludens@math.klte.hu

Burnard, Lou
Oxford University, UK
lou.burnard@computing-services.
oxford.ac.uk

Bzyl, Włodzimierz
University of Gdańsk
Gdańsk, Poland
matwb@julia.univ.gda.pl

Chandy, Yakov
Chennai, India
yakov@tnq.co.in

Christiansen, Kaja
Computer Science Department
University of Aarhus
Århus, Denmark
kaja@daimi.au.dk

Esfahbod, Behdad
Computing Center
Sharif University of Technology
Tehran, Iran
behdad@bamdad.org

Feng, Hong
Wuhan, Hubei Province
430040 China P.R.
hongfeng@gnu.org

Gaudeul, Alexandre
Toulouse, France
Alexandre.Gaudeul@
univ-tlse1.fr

Ghosh Dastidar, Prabir
ICMAM, NIOT Campus
Chennai
Tamil Nadu, India
prabirgd@rediffmail.com

Grathwohl, Steve
Duke University Press
Durham, NC, USA
grath@duke.edu

Guravage, Michael A
National Aerospace Laboratory
Amsterdam, Netherlands
guravage@nlr.nl

Housley, Brian
GCCS GmbH
Basel, Switzerland
brian.housley@gccs.ch

Jadhav, Narendra
Belapur, New Bombay, India
nagesh@dncdata.com

Jaiswal, Deo Kishor
Bangalore, Karnataka, India
kishor@srnova.com

Joshi, Manjusha
Pune, India
manjushasjoshi@hotmail.com

Kalidoss, Murugesh
St. Joseph's College
Trichy, India

Kastrup, David
Kriemhildstr. 15
44793 Bochum
Germany
David.Kastrup@t-online.de

Krishnan, E
University College
Trivandrum, India
Ekmath@md5.vsnl.net.in

Kuester, Johannes
Typoma
Holzkirchen, Germany
jk@typoma.com

Kumar, KG
Pattom
Trivandrum, India
Kg@myiris.com

Kumar, Suresh
New Delhi, India
skumar@techbooks.com

Kurup, VU
Media Mates
Trivandrum, India
Kurupvu@hotmail.com

Levine, Jenny
Duke University Press
Durham, NC, USA
jenny.levine@duke.edu

Mathurbutham, Sankaran
Bangalore, Karnataka, India
sankaran@srnova.com

McKay, Wendy
California Institute of Technology
Pasadena, CA, USA
wgm@cds.caltech.edu

Moore, Ross
Macquarie University
Sydney, Australia
ross@maths.mq.edu.au

Nagarjuna, G
Free Software Foundation of India
India
Nagarjuna@gnu.org.in

Nambooripad, KSS
Center for Math Sciences
Trivandrum, India
Kssn@md2.vsnl.net.in

Pai, Nagesh
New Bombay, India
nagesh@dncdata.com

Pal, Palash P
Saha Institute of Nuclear Physics
Calcutta, India
pbpal@theory.saha.ernet.in

Paul, Stephen
New Bombay, India
nagesh@dncdata.com

Phadke, SB
Institute of Armament Technology,
Girinagar
Pune, India
sbphadke@hotmail.com

Píška, Karel
Institute of Physics
Academy of Sciences
Prague, Czech Republic
piska@fzu.cz

Plaice, John
School of Computer Science and
Engineering
The University of New South
Wales
UNSW Sydney NSW 2052,
Australia
plaice@cse.unsw.edu.au

Popineau, Fabrice
SUPELEC
57070 Metz, France
Fabrice.Popineau@supelec.fr

Pournader, Roozbeh
Computing Center
Sharif University of Technology
Tehran, Iran
roozbeh@sharif.edu

Prakash, NA
Indian Academy of Sciences
India
prakash@ias.ernet.in

Radhakrishnan, CV
Focal Image (India)
Trivandrum, India
Cvr@river-valley.org

Rahtz, Sebastian
Oxford University Computing
Services
Oxford, UK
sebastian.rahtz@oucs.ox.ac.uk

Rajagopal, CV
Focal Image (India)
Trivandrum, India
Cvr3@river-valley.org

Rajkumar, Ebenezer J
Bangalore, India
rajkumar@srnova.com

Rajkumar, S
Linuxense Information Systems
Trivandrum, India
Raj@linuxense.com

Ranade, Ajit
Bombay, India
ajit.ranade@in.abnamro.com

Ravoor, Nijalingappa
Bangalore, Karnataka, India
ravoor@macindia.soft.net

Rodionov, Igor
Department of Computer Science
University of Western Ontario
London, Canada
igor@csd.uwo.ca

Roegel, Denis B
LORIA
Campus scientifique
54506 Vandœuvre-lès-Nancy cedex
France
Denis.Roegel@loria.fr

Rowley, Chris
Faculty of Mathematics and
Computing
Open University
London, UK
c.a.rowley@open.ac.uk

Sangeetha, G.S.
Bangalore, Karnataka, India
sam@macindia.soft.net

Sasirekha, T
Bangalore, Karnataka, India
sasirekha@macindia.soft.net

Satish, Babu
InApp Pvt Ltd.
Trivandrum, India
Sb@inapp.com

Schaa, Volker RW
DANTE e.V.
Germany
president@dante.de

Sebastian, Johny
New Delhi, India
jsebasti@techbooks.com

Shukla, Wagish
Indian Institute of Technology
New Delhi, India
wagishs@maths.iitd.ernet.in

Skoupý, Karel
Zürich, Switzerland
skoupy@inf.ethz.ch

Smirnova, Elena
Department of Computer Science
University of Western Ontario
London, Canada

Smith, Alistair
Sunrise Setting
Torquay, UK
alistair.smith@btconnect.com

Swaminathan, Swarnalatha
Chennai, India
latha_goutham@vsnl.com

Talole, SE
Institute of Armament Technology,
Girinagar
Pune, India
setalole@hotmail.com

Thakur, Samanta
Belapur, New Bombay, India
nagesh@dncdata.com

Trehan, Amitabh
Mahatma Gandhi Antarrashtriya
Hindi Vishwavidyalaya (MGAHV)
New Delhi, India
amitabhtrehan@yahoo.co.in

Tripathi, Sanjay
Bangalore, Karnataka, India
hrd@srnova.com

Venkatesan, SK
Chennai, India
skvenkat@tnq.co.in

Watt, Stephen
Ontario Research Center for
Computer Algebra
University of Western Ontario
London, Canada
Stephen.Watt@uwo.ca

Wujastyk, Dominik
University College London
London, UK
ucgadkw@ucl.ac.uk

Kaveh and Sebastian taking the group photo

TEX and Databases – TEXDBI

K. Anil Kumar
Linuxense Information Systems Pvt. Ltd.
Trivandrum, India
http://www.linuxense.com
anil@linuxense.com

Abstract

Report generation is one of the demanding areas of enterprise computing. It involves complex database queries, drawing conclusions, making projections and presenting them in an easy-to-comprehend manner. This paper describes how to use TEX to deal with the presentation aspects in a productive way.

Introduction

TEX is highly programmable and has a high degree of consistency in maintaining the structure and layout of the documents generated. Moreover, no other typesetting system can claim the precision control of typesetting parameters that TEX does. This makes TEX a good tool to prepare reports both for printing and for Web publishing. However, TEX lacks one capability as a report generation tool: it cannot directly communicate with external systems like database engines. Mostly data for reports reside in database systems and must be retrieved based on rules. So we are compelled to depend on other technologies and systems to achieve this. This may also mean bringing in additional people to get things done.

First, let us have a look at various techniques for generating reports using TEX which are in use today.

Ways to Generate Reports Using TEX

We have been using TEX to generate reports using a variety of techniques. And these techniques can be classified into two categories: 1) generate TEX documents using other languages; 2) embed other languages in TEX documents and use a preprocessor. Both of these techniques are explained below.

Use Database-enabled Languages to Generate TEX Documents A Java (for example)[1] program can create a TEX file with the data retrieved from a database. We then compile this TEX code to get a presentable report. The process contains the following steps:

1. A TEX programmer prepares a TEX document that we will call a template. This template contains no useful information itself, though it is a complete TEX document as far as TEX is concerned. We add the specific data to it, e.g., it may be a table construct defined with all the required parameters, but no actual rows of data. The template only becomes a useful document when the data is added.
2. This TEX template is handed over to the Java programmer to write a program, with this template hard-coded within it, which will in turn write a TEX document with the data retrieved from a database engine.
3. The generated TEX document is compiled to get the report.

There is one disadvantage with this approach. It is difficult for a TEX programmer to intervene to make a change in the report format (i.e., in the template) after step 2 given above. This is because the TEX code has already been embedded into the Java program in a very different form, which doesn't make any sense to a TEX programmer at all. There are only a couple of solutions to tackle the situation:

1. Repeat step 1 and 2 with the modified format specification; or
2. Have the TEX programmer and the Java programmer work together on making the required changes.

In a production environment, where work must be done very quickly, neither of these approaches are feasible. To repeat the process all over again is definitely a bad idea and bringing two groups of people, as suggested in the second solution, to work on a problem is difficult to manage and time-consuming.

[1] The language could be Perl, C++ or any other with the capability to interact with a database engine.

Embed Other Languages in TEX There is a better approach than this first one. We can embed statements in a database-capable language in the TEX template. Then through a pre-compilation process we can execute those embedded code snippets and replace them with the data retrieved from the database to make a "data-filled" TEX document.

Let us analyze the steps involved in generating a report this way.

1. As before, a TEX programmer prepares a template.
2. This template is handed over to the programmer who will embed code snippets required to retrieve data from the database and fill-in the template.
3. The template undergoes pre-compilation and the resulting TEX document is compiled. The report is ready.

In this approach, even after embedding the code snippets the TEX template will look like a regular TEX document and a TEX programmer can still modify it if there is a specification change. So there is no need to bring in the other programmer to make a modification in the report presentation.

Pre-compilation is the major disadvantage of this approach. Every time to generate a report one has to pre-compile the document and then run the TEX compiler to get the report. Moreover, the TEX coder either has to learn a database capable language or he/she has to depend on someone who knows it.

Enable TEX to Communicate with External Systems This approach will enable TEX to communicate with external systems like database engines to retrieve information to typeset reports directly. There is no need to depend on other languages and there is no pre-processing involved. This approach has two advantages:

1. A TEX programmer can prepare reports by himself/herself without depending on another language programmer.
2. Development cycle is faster. Changes can be incorporated in the TEX document directly and run the TEX compiler; and it is done!

How to Enable TEX to Talk to Database Engines?

A running TEX compiler is a process[2]. For a process, communicating with the external world means communicating with other processes, either running on the same computer or on a different computer in the network. TEX has no built-in interprocess communication facility but they are commonly provided via the following mechanisms:

1. Writing to a file using `\write16`.
2. Executing a shell command using `\write18`.
3. Reading in a file using `\input`.
4. Writing to terminal using `\write15`.

We are limited to these mechanisms for I/O capabilities with TEX.

Any modern operating system supports named pipes[3] and sockets for communication between two arbitrary processes. Files are also a way of passing data to the external world. However, considering the I/O capabilities of TEX, communication with sockets does not seem feasible and so the option is either files or named pipes.

Files or Named Pipes? We can make TEX to write to a normal disk file and instruct an external system to read from it. This is a simple way of exchanging data between two arbitrary processes. It is quite simple to understand, and very easy to implement. But the downside is that it is very difficult to synchronize communication and it can become out of control in certain situations. Named pipes or FIFOs are better in this area.

Named pipes, also known as FIFO structure, can be effectively used with TEX for interprocess communication. For TEX, a named pipe will appear as a regular file. When TEX writes to this file, the operating system stores the data in a buffer and then it can be read by the other program. For the reading program, data will appear in the order it was written. Also the reading process will be blocked till the other party writes to it. Similarly, the writing process will be blocked until the other process starts reading it [4]. This will provide the required synchronization for the "conversation."

Hence TEX can write and read from a named pipe to talk to another process and that process can do the same. Thus, it becomes possible for TEX to communicate with another process and the architecture discussed in this paper for database communication is built upon this idea.

[2] That is, an operating system process which can be identified by a process ID.

[3] A named pipe, also known as FIFO (first-in-first-out), is similar to device files in Unix/Linux. They have a *disk inode* (and a file name) and hence can be accessed by any process. As the name implies, with FIFO the first byte written into it will be the first byte read from it.

[4] Here we assume that the named pipe is opened in *blocking* mode. If the named pipe is opened in *nonblocking* mode the read will return whatever bytes are available (perhaps none).

A TEX Abstraction for Database Communication: A Middle Layer

Interprocess communication and named pipes are far beyond the interest of a TEX programmer and it is definitely a bad idea to suggest a TEX programmer to use all these things while doing TEX coding! So a good implementation of this idea should provide a TEX-like abstraction of the mechanisms discussed above. This abstraction should have the least possible learning curve and should adhere to standard TEX coding conventions.

An *n*-Tier Architecture Here we are going to make TEX to talk to a database engine. At the top there is the TEX compiler and at the bottom there runs a database engine. And in between we introduce two components: one component gives a TEX abstraction of the systems beneath and the second component bridges the TEX abstraction and the database engine. These two components together are called TEX-DBI.

TEX Abstraction of a Database Operation

Querying a database involves opening a connection to the database, passing an SQL query, retrieving the result and finally closing the result object and then the database connection. A LATEX package can be used to define a set of macros to do all these jobs on behalf of the TEX programmer.

Following are the macros that we defined in the implementation of this idea.

```
\begin{tex-dbi}[host=xx.yy.zz,%
  dbname=sampledb,uname=anil,passwd=mypw]
```

This will initialize a database transaction for the session[5]. There can be more than one transaction per session but they are not supposed to overlap. Here is a description of the parameters:

Parameter	Meaning
`host`	Name of the computer running the database engine
`dbname`	Database name
`uname`	Database username
`passwd`	Database password for the specified username

```
\texdbiexec{"select * from employee"}
```

This macro passes the SQL statement to the underlying database engine. The TEX macro is not responsible for checking the accuracy of the SQL given; it just passes the statement to the underlying system.

```
\texdbicount{}
```

This macro returns the number of rows (possibly zero) resulting from the query. This value may be used as the limiting value for a loop or just to check whether the query returned successfully.

```
\texdbinext{}
```

The result object exposes only one row at a time of the possibly several rows returned by the query. This macro moves an imaginary pointer through the result set, making each row current in turn. When a new result is obtained the imaginary row pointer doesn't point to any row; a call to this macro then sets the pointer to the first row returned.

```
\texdbivalue{field_name}
```

This macro returns the specified field of the current row. `field_name` should match the name of the field specified in the SQL statement[6].

```
\end{texdbi}
```

This macro ends a database session. The connection will be closed and the underlying component will be ready to start a new transaction.

Normally, these macros are called in the sequence given above. `\texdbinext{}` can be called as many times as the number returned by the macro `\texdbicount{}`. `\texdbivalue{}` can be called with appropriate parameters as many times as required after each call to `\texdbinext{}`.

The Bridging Component The TEX macros described in the above section communicate with this component through a named pipe as described earlier. It is called a bridging component because it connects TEX and the database engine. The communication between the bridging component and the database engine itself happens through a TCP/IP or Unix domain socket.

The request in the TEX code will be passed to the bridge by the macros through the FIFO and these requests are translated to equivalent JDBC or ODBC commands (or in some other format required by the mechanism being used by the bridging component) as the case may be. The results from the database engine obtained by the bridge is written back to the FIFO for the TEX macros to read.

[5] Or current pass of compilation.

[6] This rule is enforced by the underlying bridging component which makes use of JDBC, ODBC or similar mechanisms to transact with a database engine, as described next.

Adding Session Capability to Make the Middle Layer a True Multi-user System

It is also possible for this system to support sessions, thus making it a true multi-user system. A Web server session-ID-like mechanism can be employed to identify each request and thus to communicate with multiple TEX compilation sessions simultaneously.

An alternative to this session-ID approach is to use session-specific FIFOs: TEX macros can create a FIFO through a shell command and the name of the FIFO can be passed to the bridging component as part of the transaction initiation request. The rest of the transaction can happen though that FIFO.

If multiple concurrent sessions are supported, only one middle layer is enough to support multi-user enterprise needs.

Conclusion

Enabling TEX to communicate with a database engine will eliminate the need to involve other languages to generate quality reports. A TEX programmer can easily learn and use the TEX macros defined by the TEX-DBI system. Development life-cycle will be shorter and the process will be completely in the hands of a TEX programmer.

An implementation of this concept can be found at: `http://www.linuxense.com/oss/texdbi/`.

References

[1] Knuth, Donald E., 1986, *The TEXbook*, Addison-Wesley.

[2] Bovet, Daniel P. & Cesati, Marco, 2001, *Understanding the Linux Kernel*, O'Reilly.

New Horizons of Free Software: An Indian Perspective

Satish Babu
Vice President (Southern Region)
Computer Society of India
sb@inapp.com

Abstract

Free Software has been quietly but steadily making inroads into the world of software. While this process is important, nowhere are the ramifications potentially so productive as in a country like India. India is rich in programmer resources, but is otherwise poor. India's governments, institutions and the industry require a huge quantity of software, but due to a variety of reasons—including intellectual property rights and costs—it had been hitherto not possible to produce software for these requirements. These limitations are adequately addressed in the Free Software model, making it a key enabler in the development process for India.

Introduction

Information and Communication Technologies (ICTs) have been a mixed blessing for many Third World countries. On the one hand, ICTs bring a number of immediate and perceivable benefits to most societies. On the other hand, they result in dependencies on external entities over which these countries have little or no control. While it is clear that ICTs are necessary for a country to step into the third millennium, it is less clear what the specific problems are—for the day and for tomorrow—that these technologies bring in.

One of the most enduring artifacts of ICT is software. Endowed with several unique characteristics, software epitomizes the unique combination of power, efficiency, and opportunities of ICT. Some developing countries such as India have been able to make a global impact by its software strengths—essentially its large pool of trained human resources. Others—African countries, for instance—have been marginalized on account of a dearth of trained manpower.

The last two decades have seen the emergence of a few software superpowers—giant global corporations—that have cornered a significant proportion of the world's software market. These corporations have effectively utilized the proprietary model of software development—where the software is *owned* by the corporation—while the user only purchases the right to *use* the software (that too for a limited period in some models).

Even while many nations have anti-monopolist measures for tangible goods, the intangible nature of software has meant the absence of such safeguards of the rights of citizens. As information monopolies continue the consolidation of their market segments, many societies, people, and countries find themselves at a loss how to defend the interests of their own constituencies.

Software Alternatives

The proprietary model of software development, although the dominant model today, is by no means the only successful paradigm. The Free Software mode of software development provides a countervailing paradigm that is both increasingly visible and extremely successful. The chief differentiating factor of the Free Software mode of software development is that Free Software is always accompanied by its source code and has a licensing scheme that is usually a variant of the GNU General Public Licence (GPL), which together confer a set of powerful benefits to the user.

Much confusion has arisen from the use of the word 'free', and much has been made of the distinction between free beer and free speech. Thankfully, this confusion doesn't exist in most non-English languages, where '*software libre*' has now become both visible and easier understood.

Free Software (used here in the broadest sense, and inclusive of the 'Open Source' label as well) empowers the user, providing her with a variety of 'freedoms', including the freedom to use the software, study it, modify it, and redistribute it, as well as guaranteeing that the product can never lose these freedoms.

While these freedoms are themselves extremely important, their collective import, especially for larger organizations, including governments, is far reaching. This paper examines these vis-á-vis developing countries in general, and India, in particular.

Developing countries share several attributes: poverty; lack of education; opaque governments; malnutrition and epidemics; and poor respect for human rights, to list some. While the well-known aphorism, "you can't eat software", is indeed true, it is increasingly being recognized that ICTs can play an important role in addressing development priorities and can improve the quality of life of some of the poorest sections of human society. This paper argues that *software libre* is in a position to make this happen, and is perhaps the only way to protect the rights of peoples and communities, for today and tomorrow.

The Ethical Dimension

Humankind has, in the centuries of its evolution, come up with several universal ethical values: truth, beauty, freedom, courage, justice, peace and harmony. Of these, the one that has most influenced the destinies of nations is freedom. Freedom has been described variously as "the power to act, speak or think without externally imposed restraints" or as "the right to choose".

It is interesting to note that while countries, in general, are enhancing freedoms available to its citizens (improving human rights; supporting democracy as the nearest-to-ideal for governance), one glaring exception, where freedom is actually being curtailed, relates to software. The emergence of information monopolies not just limit choice, but also tend to stifle the emergence of alternate paradigms. In software, where the marginal cost of reproduction approaches zero, there is a clear risk of larger companies imposing 'invisible' barriers against alternatives (for example, by using private application programming interfaces (APIs) not known to competitors, or evolving custom 'extensions' to commodified standards, thereby limiting the users' freedom of choice).

Software libre provides a truly 'free', morally desirable alternative, by making it possible for alternative products and services to emerge, even where they would be infeasible from purely profit oriented market dynamics. This is, of course, not accidental: Free Software has evolved to be a framework of complementary set of software building blocks that are available at low- or zero-cost, thereby making it possible for the further development in the same manner. In short, *the freedoms enshrined in Free Software, together with its much lower costs, allows it to break the profitability barrier, thereby allowing alternative products to emerge even where market dynamics dictate otherwise.*

The Legal Political Dimension

From a pragmatic sense, the most important point of distinction between proprietary and *software libre* has to do with its politico-legal aspects. First, Free Software removes much of the licensing nightmare associated with proprietary software (for example: per copy, per seat, per CPU, timebased activation), and actually 'liberates' software as also the users. Second, the freedoms associated with Free Software becomes tremendously important in some contexts—e.g., for a government, on account of factors such as sovereignty, autonomy and information security. Third, it allows for modification and maintenance by any qualified entity, thereby avoiding the vendor 'lock-in' associated with most proprietary software. Finally, Free Software allows new software to be evolved or modified from earlier software, reusing code where required with the guarantee of inalienable rights to the fully developed artifact.

For India, this dimension perhaps overshadows most other advantages of Free Software. Governments are taking steps to implement IT enabled services in India's provinces. These efforts do not, in most cases, fully take into account the ramifications of such government decisions. At least in some cases, intellectual property related disputes with local companies have already come to light. One wonders how these governments—which are presently finding it difficult to deal with small local companies vis-á-vis licensing issues—will be able to deal with the giant software global corporations and yet ensure that the interests of its own people are protected.

As software entrenches itself as a prime enabler of governance, it is extremely important for governments to understand this dimension. The *software libre* model is the only solution that will protect the rights of governments and its people. The following quote, reproduced from *A Rebuttal to Meyer's "The Ethics of Free Software"*, available at `http://www.advogato.org/article/94.html`, makes this point amply clear:

> Having the source is about people, the everyman and everywoman, keeping a stake in the software technology that's fusing with, running and controlling their everyday lives. It's not about exercising power, it's about the freedom and ability to exercise that power when needed.

The Technology Dimension

The foregoing discussion on ethics or legal aspects of *software libre* would have been only of academic interest if the products of the model were merely as good as those of the proprietary model. However, it can safely be said now that the genre of software produced by *software libre* is thought (by programmers, and, lately, by the industry as well) to be technologically superior.

User perception of software quality is dependent on a number of parameters: performance; features; the presence and number of bugs; presence of security holes; ability to incorporate users' own requirements; availability of patches and fixes; and the availability of a community of users who can help novices.

Software libre scores high on almost all of these metrics. It is interesting to note that the proprietary stream highlights the 'fact' that "Free Software is unsupported". Even when legally and technically correct, the statement is especially ironic in the context of Southern countries, including India. While users pay the same price (in most cases) for products in the South and the North (although, by Human Development Index standards, the relative price is many times more in the South than in the North), the 'support'—which has already been factored into the price—is virtually nonexistent in Southern countries (where telephone infrastructure itself is nonexistent in many situations). Most people who buy shrink wrapped software in India *never use telephonic support.*

Other observers have pointed out how the superior quality of *software libre* is not an accident. On the contrary, this superiority has to do with its development methodology, which features the following practices:

- Large numbers of programmers involved
- One or more maintainers, who alone can change code in the repository
- Peer review of code
- Parallel testing/debugging by numerous users
- Bug reporting by hundreds of users
- A sense of ownership of the code
- Recognition to chief contributors in documentation

Security by obscurity, long promoted by proprietary companies, is clearly a very poor substitute to parallel testing and peer reviews, all by a community of programmers who feel a sense of ownership in what they do.

The Economic Dimension

If the fruits of ICT should benefit a significantly large section of humankind, some kind of a democratization of computing ('computing for the masses') is required in Southern countries. There are several obstacles to such an initiative, the most important of which are the lack of funds for hardware and software, and the lack of basic infrastructure. While the latter is outside the scope of this paper (although this is being addressed by development programmes based on both internal resources and external aid), the former is something that can be addressed by *software libre.*

While free as in free speech is very important, almost equally important for the South is free as in free beer. The relatively low cost of *software libre* products—for now and for the future—is an important consideration that can influence the decision to adopt it. While exact figures for the savings by using Free Software vary, what is clear is that there *are* substantial savings, even when the relatively large costs of creating trained manpower for free platforms (which may not exist in most places) are factored into the analysis.

It is important to realize that the 'price' of software is different from its cost. Several proprietary companies are known to offer substantial discounts on the list price of their software, especially to institutional buyers such as governments or schools. While this results in a temporary reduction in the funds outflow, there is no guarantee that—once the lockin has happened—the vendor will not increase prices. *The cost of proprietary software is high, no matter what its price: freedom has been lost.*

Other Dimensions

Software libre is socially desirable from several other aspects as well. Some of these are examined below:

Education India has an edge over most other countries in IT, since it has a large pool of trained human resources in the area. In order for India to continue to maintain this edge, it is important to ensure that its students have access to the latest technologies and practices in IT. Free Software *is perhaps the only example of live code that is available for students to examine.* No surprise that many premier institutions in India already use such source code for teaching their students.

Globalization and Market Penetration While some communities are able to utilize globalization to further their own interests, most communities in the South are vulnerable—through loss of livelihoods,

and markets—to the relentless process of globalization.

Globalization brings with it the tendency for businesses to shift to globally competitive locations. It is difficult to resist this process, as the rubber and coconut farmers in the Kerala province of India have realized to their dismay.

It is important to empower smaller formations (microcompanies, programmer cooperatives) that can provide localized solutions to local problems of communities where possible. While globalized solution development (for both free and proprietary models) will continue to exist, micro-initiatives will surely have a role in contributing to self-sufficiency and autonomy of smaller communities scattered over the globe. *Software libre* can be an important enabler of this process.

Cultural Diversity That the world faces the risk of a 'global monoculture'—where a specific culture emerges as globally dominant—and that IT in its present form would contribute towards this process, has been pointed out by several researchers. The absence of localization of software has been raised as one factor that contributes to this process, as has been reported from Iceland.

In India, where over 500 languages are spoken, and where there are over a dozen distinct scripts, the problem of preservation of cultural diversity assumes importance. From direct experience, proprietary software vendors will develop localized versions if—and only if—this move makes commercial sense. If not, the community is doomed never to use computers in their own language. This is the reason why in Kerala one of the most eagerly awaited developments is that of a Malayalam font on *software libre* platforms (work for which is under way at present).

Gender Women form one of the most vulnerable sections of most Southern communities, having to bear, as they do, the additional burdens of cultural taboos, childbearing and weaning, and minding the household. All this has meant that, by and large, the average Southern woman is far removed from computers and the Internet.

There are, however, many initiatives that are trying to bring women into the technological mainstream, generally in the form of IT enabled services. The *Kudumbasri* experiment in Kerala, for example, organizes women to carry out data entry operations. While there is an element of subsidy to these organizations at this time, they will definitely need to become financially viable within a short time.

For organizations of women, as well as for individual women, *software libre* could prove to be a true liberator—it doesn't carry any licensing penalty, is extremely cost effective, is lightweight, is easy to locate and download (as opposed to purchasing a product, and getting it installed by an external technician), and is free of unnecessary features that make some proprietary software overly complex. It would not be an exaggeration to say that *software libre*, in general, is more woman friendly than proprietary software.

In Conclusion: From Subaltern to Mainstream

Software libre has moved from the fringes of computing to its mainstream. The values underpinning it have provided it with a vision that is only now—nearly 25 years after its founding—being realized in its entirety.

This paper has argued that for national governments and concerned individuals alike, *software libre* is a better option on account of its inherent ethical vision. For national governments, *software libre* protects their autonomy and sovereignty, whereas for the individual, it provides the freedom to choose. For enterprises and organizations, *software libre* provides a pragmatic balance between economics and functionality.

References

[1] Goerzen, John, *The Ethics of Free Software*, `http://www.complete.org/papers/fsethics/`

[2] Perens, Bruce, *Why security through obscurity doesn't work*, *Slashdot*, July 20, 1998 `http://slashdot.org/features/980720/0819202.shtml`

[3] Stallman, M. Richard, *Why software should be free*, November 1, 1998, `http://www.fsf.org/philosophy/shouldbefree.html`

[4] Xiphmont, *A rebuttal to Meyer's "The Ethics of Free Software"*, May 21, 2000, `http://www.advogato.org/article/94.html`

On the Localization of TEX in Hungary

Gyöngyi Bujdosó
Department of Computer Graphics and Library and Information Science
Institute of Mathematics and Informatics
University of Debrecen
H-4010 Debrecen, P.O.B. 12
Hungary
ludens@math.klte.hu

Ferenc Wettl
Department of Algebra
Institute of Mathematics
Budapest University of Technology and Economics
Budapest, Műegyetem rakpart 1–3, H. ép. V. 5.
Hungary
wettl@math.bme.hu

Abstract

This paper deals with the present and future of the localization of TEX in Hungary. The authors review some of the necessary tools for preparation Hungarian documents, and especially the improvements needed to make TEX more usable in Hungary. Some of the work has been done, and a short "to do" list will be presented for work to be done in the near future. The problems stemming from the specialities of Hungarian grammar (e.g., hyphenation, handling definite articles and suffixes) will be considered as well as the tasks implied by the heritage of the Hungarian typography (e.g., page layout).

Introduction

The Hungarian Users Group (called MATEX) exists formally since the end of 2001. As the very first activity, we have resumed the localization of (LA)TEX for the Hungarian language. We have started getting together all the developments in the domain of localization, we are looking for the problems which have not been solved, and trying to organize teams for finding out the answers.

This paper deals with the developments which have been achieved and with our aims for the near and distant future.

Grammar

There are some specialties in the Hungarian language which might be interesting in connection with TEX or generally with document preparation. We concentrate on the problems of generated texts.

Definite articles In Hungarian there are two definite articles, 'a' and 'az'. 'a' is used before words beginning with a consonant, and 'az' is used before words beginning with a vowel, just like 'a/an' in English. If any generated text needs an article, it must also be generated. This is the situation with `\ref`, `\pageref`, and `\cite` in LATEX. The `babel` package nicely solves this problem with a more generally usable command `\az`. This generates the definite article along its argument and the argument itself; that is, the command `\az`{⟨*arg*⟩} is equivalent either to `az~`⟨*arg*⟩ or to `a~`⟨*arg*⟩ depending on the first letter of ⟨*arg*⟩. Beside `\ref`, `\pageref`, and `\cite` one may use `\aref`, `\apageref`, and `\acite` with `babel/magyar` which also generate the appropriate definite articles. These macros use the command `\az`. The rule mentioned above has some consequences, which are satisfied by `\az`:

- a number beginning with 5 or a number beginning with 1 and having $3k + 1$ digits (k is a nonnegative integer) is preceded by 'az', all the other numbers are preceded by 'a'. For example 'az' is before 1, 5, 51, 524, 1020, 1000000, and 'a' is before 2, 3, 4, 6–49, 60–499, 10000, 100000, etc.
- the same rule is applied to roman numerals; that is, 'az' is before I, V, LI, DXXIV, MXX, and 'a' is before II, III, IV, VI, etc.

- if a notation has one letter only, or begins with a letter and is followed by a number or any special character, then the pronunciation of the letter must be considered. The pronunciation of 'f', 'l', 'm', 'n', 'r', 's', 'x', 'y' begins with a vowel. For example 'az F. fejezet' (chapter F), 'az x1 változó' (variable x1) is the correct form.

There were several contributors to the Hungarian part of babel. We would especially like to highlight the work of József Bérces and of course that of Johannes Braams [1].

Alphabetical order The special rules of alphabetical ordering are as follows:

- A one character consonant is handled separately from a two character consonant beginning with the same sign. For example, 'c' and 'cs' are two different consonants, so 'cukor', 'cuppant', 'csalit', 'csata' is the correct order. This rule is not applied for the ancient type of two character letters, which are frequently used in family names, and for the two or more character letters of other languages, like 'sch'. This rule can be well handled by the xindy package [4]. This and some of the following problems can be solved with the makeindex package [6] supplemented with the application of an extra script/preprocessor which applies the '@' metacharacter in the index entry.
- The short and long vowels are equivalent (a=á, e=é, i=í, o=ó, ö=ő, u=ú, ü=ű), although the long ones are after the short ones in the Hungarian alphabet (a, á, b, c, ...). For example 'alma', 'álom', 'alorvos' is the correct order. The only exception is the case when two words differ only in the length of the same vowels. In this case the short vowel comes first (e.g., 'kerek', 'kerék', 'kérek').
- The long digraphs are considered as two short digraphs, so the next substitutions must be applied before the ordering: 'ccs' → 'cs + cs', 'ggy' → 'gy + gy', 'ssz' → 'sz + sz', 'zzs' → 'zs + zs', etc. This is also true for the only trigraph 'dzs', where the substitution is 'ddzs' → 'dzs + dzs'.

Inflexional suffixes There are several things to do in connection with TEXing in Hungarian. The topic of this section is not the most urgent or important, but it is interesting in general as a question of the generated texts in Hungarian documents. The essence of the problem is that there are several suffixes, and most of them have more then one form.

Let us suppose that some equations are numbered from (1) to (7) in a math text. The translation of the English text "adding (1) to (3) and subtracting it from (4) gives (5)" is "(1)-et hozzáadva (3)-hoz, majd azt kivonva (4)-ből az (5)-öt kapjuk." If we change the numbers only, we may get different suffixes as the next example shows: "(3)-at hozzáadva (4)-hez, majd azt kivonva (6)-ból a (7)-et kapjuk." The problem with such a sentence from TEX's point of view is that generating the equation numbers also demands generating the suffixes. The suffixes follow the vowel harmony. This means that suffixes, which may assume two or three different forms, usually agree with the last vowel of the stem. In other words, front vs. back alternatives of suffixes are selected according to the vowel(s)[1] the stem contains [8]. Examples: 'tűzből' (from fire), 'házból' (from house), where 'tűz' and 'ház' are the base words and the front and back forms of the suffix are '-ből' and '-ból'.

If the suffix has three forms, one of them has a back vowel (o), the other has a labial front vowel (ö), and the third has an illabial front vowel (e). If the last vowel of the stem is labial (illabial), the labial (illabial) suffix is used. Harmony causes the following alternations among suffix combinations:

- a/e (-ban/-ben 'in', -nak/-nek 'to'),
- á/é (-nál/-nél 'at'),
- ó/ő (-ból/-ből 'from', -ról/-ről 'about'),
- u/ü (-ul/-ül 'for, by'),
- o/e/ö (-hoz/-hez/-höz 'to').

There are several uncertainties. For example the vowels may show paradigmatic alternations as long and short vowels alternate in some stems (ver*é*b → ver*e*bet 'sparrow', f*a* → f*á*t 'tree'). Another problem is that 'i' and 'í' can be both front and back harmonic (híd → hidat 'bridge' + acc., but szív → szívet 'heart' + acc.). The suffix may have different forms if the word is a compound word. So a good suffix-generator needs a dictionary.

As the first example shows, it can be advantageous to solve the problem for numbers. Fortunately, the suffix depends on the last nonzero digit and on the number of closing zeros only:

- back harmonic numbers: 0, 3, 6, 8, 100;

[1] The vowels created in the front of the oral cavity are called *front* vowels, and those formed in the back of the oral cavity are called *back* vowels. The front/back vowels cause the feeling of high/low sound. In Hungarian the front vowels are e, é, ö, ő, ü, ű, the back vowels are a, á, o, ó, u, ú. The vowels i and í are neutral as they can be either front or back vowels depending on the word. For example í is a front vowel in the word 'víz' (water), but a back vowel in the word 'zsír' (grease). Four of the front vowels are labial (ö, ő, ü, ű), others are illabial (e, é, i, í). A suffix is called front (back) suffix if the vowel it contains is a front (back) vowel.

- labial front harmonic numbers: 2, 5;
- illabial front harmonic numbers: 1, 4, 7, 9, 10, 1000.

These grammatical problems also come up in relation to spell-checking. Until recently, only commercial programs were available; happily, the first Hungarian GNU ispell program [10] and a free Linux and FreeBSD version of a commercial program were released recently [9].

Hyphenation The limitations of TEX in hyphenation of non-English text are well known. Unfortunately some of the problems are still not solved, or in some cases the known solutions cause other problems. The phonetic rules of Hungarian hyphenation are simple and easily programmable (unfortunately in TEX the third doesn't):

- Every syllable has exactly one vowel, so a Hungarian word has as many syllables as vowels (fi-a-i, me-ta-fo-ra, pa-ra-di-csom).
- A chain of consonants between two vowels is cut before the last consonant, so the last one starts the next syllable (csu-por, kap-tár, Hamburg-ban). In Hungarian 'cs', 'dz', 'gy', 'ly', 'ny', 'sz', 'ty', 'zs' are digraphs, and 'dzs' is a trigraph, that is, two or three letters form one consonant (ki-csi, gú-nya).
- Although only the first letter is doubled in a long digraph (or trigraph), when hyphenated both syllables contain the full digraph (or trigraph) (mennyi – meny-nyi, hosszú – hosz-szú, gallyak – galy-lyak, briddzsel – bridzs-dzsel)

There is a grammatic rule of hyphenation, which overrides the previous phonetic ones and causes real difficulties:

- Compound words or words beginning with verbal or superlative prefixes have to be hyphenated at the morpheme boundaries (ö-reg-asz-szony – öreg+asszony, meg-e-szi – meg+eszi).

To handle this, either a (never complete) exception list or a morphological analysis is needed. At the moment TEX uses the first method (the wordlist is implicitly given in `huhyph.tex`), but only the second can produce a perfect solution. We list some cases when the fourth rule conflicts with the first three ones:

- Two words, a simple and a compound or prefixed, have the same form but different hyphenation (fe-lül – over, fel-ül – sit up, me-gint – again, meg-int warn, gép-e-lem – machine part, gé-pe-lem – I type it).
- Morpheme boundary seems to be a long digraph (villamos-szék – electric chair).
- Either of two hyphenations is acceptable if there is a Latin or Greek morpheme boundary, but it is not clear for the average reader (depresszió dep-resz-szió or de-presz-szió – depression).

Words containing a hyphen may be hyphenated at other points according to the rules. The hyphen may be repeated at the beginning of the next line if it is necessary to show the hyphen, for example in a specialized book: nátrium-<*newline*>-klorid. More difficulties are implied by the typographic rule that no hyphenation can be applied after the first or before the last letter of a word when applied to compound words (in this case, `\lefthyphenmin` and `\righthyphenmin` can not be used).

The present official version of the hyphenation file `huhyph.tex` is made by hand, and not by `patgen`. It fulfills the first two phonetic rules by a simple list of the possible syllable boundaries, and the grammatical rule by an exception list. Recently Gyula Mayer made a big hyphenation dictionary for `patgen` [7], and generated new hyphenation patterns.

In summary, all of these problems show that joining a morphologic analyser to TEX would produce better results.

Typography of text

We would like to match the layout of texts with the Hungarian typographic traditions (see e.g. [3], [14], [15]) as much as we can. This section deals with the modifications we have to do in this field.

Baseline grid In each type of texts written in Hungarian, a baseline grid has to be applied. It is easy to typeset plain texts following this rule, but for texts containing mathematical formulae, the task is particularly difficult.

Titles It is not allowed to put a period after title names. A small typographic character on a raised position should separate the paragraph title and the text of the paragraph with a non-stretchable normal spacing around it.

> Címek ◦ Magyar nyelvű szövegekben a címek után sohasem teszünk pontot.
>
> *Jelek* ⋆ A cím betűképéhez illeszkedő bármilyen jel alkalmazható elválasztóként.

In general, the medium series of fonts is used as standard for typesetting titles within documents. The fonts can be upright, italic, in small caps or capitalized.

Application of bold upright and bold italic fonts is also allowed.

Fonts Hungarian typography generally uses bold extended fonts only for typesetting title pages, not for titles within documents (chapter, section, etc.). For these titles within documents, regular bold fonts are used.

Applying slanted fonts is absolutely contraindicated.

Short page Blank pages should not contain page numbers.

Vertical space between paragraphs The standard for every document is setting `\parskip` to 0 pt. Additional vertical space may be applied in short documents only, such as prospectus, if the paragraphs have no first line indentation.

Indentation First line indentation is the standard for typesetting paragraphs in almost every type of documents.

The indentation should be equal to 1 quad if the line length is less than or equal to 24 ciceros ([12], [13], [15]) or 20 ciceros (see e.g., [2]), otherwise the indentation should equal 2 quads. Within a given document, the measurement of the first line indentation of every paragraph (including, e.g., footnotes, references) is fixed. Every left or right indentation is equal to this size or to the multiple of it.

The first paragraph following a chapter or a section title may be typeset with or without first line indentation, but the method is fixed in a given document.

Document design without first line indentation may be applied for typesetting short documents (see e.g., [2]).

Break-line There are some rules for the length of the last line of paragraphs.

In the case of `\parindent` > 0 pt, the length of the break-line has to be longer than the first line indentation, and it has to be shorter than ⟨*line length*⟩ minus ⟨*first line indentation*⟩ or to be equal to the line length exactly.

In a document design with `\parindent` $= 0$ pt, in the case of `\parskip` $= 0$ pt, the break-line has to be longer than 1 or 2 quad and to be shorter than ⟨line length⟩ minus 2 quad (see [12], [13]) or more (see e.g., [15]). If the paragraphs are ragged right, the upper bound of the break-line length is 3/4 line length [15]. If the `\parskip` > 0 pt, the length of the break-line is allowed to be equal to the line length.

French spacing In any document, French spacing is used for spacing, i.e., the `\frenchspacing` command should be included in every Hungarian style file.

Before : ; ? ! Before some punctuation marks, such as colon, semicolon, question mark and exclamation mark, one third normal spacing should be applied.

– Megfeledkeztél a virágról?
– Nem, nem! Hoztam: ibolyát és gyöngyvirágot; kaktuszt és fikuszt; no és persze tulipánt is!

As a result, we have to modify the kerning tables of the fonts, so the standard font names used by TEX have to be changed to new ones.

Setting the spaces around exclamation marks is problematic because of its special mathematical meaning (see later).

Unnumbered lists Marks of items in unnumbered lists have to be set to a layout more closely following our traditions. The mostly used character for item labels is the en-dash. We can also use the '·', '∘', '∗' characters, and also the '•'. These latter marks should be small and on a raised position:

– Tudományterületek
  • Informatika
    ∘ Mesterséges intelligencia
      ∗ Ágensek
        · Mobil ágensek

In the typography of lists there is no vertical space between an item and the preceding or the following text (or item), nor between the paragraphs of an item. Labels are separated by two thirds normal spacing from the text.

If there is just one line per item in a list, the distance of the *labels* and the left margin of the main text is $i \times$ `\parindent` $(i = 0, 1, \ldots, 5)$.

If the items contain more than one line the distance of the left margin of the *item paragraph(s)* and the main text can be

- equal to $i \times$ `\parindent` (as it is in standard TEX list formats) $(i = 0, 1, \ldots, 5)$, or
- set to zero (i.e., just the first line of the paragraph is indented by $i \times$ `\parindent`).

Enumerated lists The numbers of items in enumerated lists have the following order and appearance:

I. Informatika
  1. Mesterséges intelligencia
    a) Ágensek
      α) Mobil ágensek

Numbers are followed by period, letters are followed by parenthesis. In the labels, letters and parentheses are emphasized.

For the rules of indentation see the preceding section.

Footnotes If a document contains relatively few footnotes (i.e., the average is less than 1.5 footnotes per page) we use asterisks (*) for marking them begining with one star on every page. Otherwise we can use numbers (as superscripts) ordinarily.

If the first line of a footnote text is indented, the footnote marks are set flush right in the space of first line indentation or in the labels of the list. The indentation in both cases has the same size as in the main text.

A lábjegyzet* jeleként a legtöbb esetben** csillagot alkalmazunk.

* Megjegyzés a szóhoz.
** Számokat alkalmazzunk, ha a szöveg valamely más részén hivatkozunk bizonyos lábjegyzetekre.

En-dash The normal usage of en-dash is the same as in English, for example, in the 'ld. 12–24. oldal' (see pages 12–24), or in the 'Budapest–Debrecen' expressions. Sometimes we are obliged to put one third (non-stretchable) normal space around the en-dash, for example, using it between names with first names: 'Kiss Előd – Nagy Pál – Tóth Éva'. The space is unbreakable before and breakable after the en-dash.

Typography of math

The Hungarian typographic traditions need some modification in the layout of mathematical formulae, too. TEXers perform some of these corrections on their own, because they see that the standard of (LA)TEX does not fit the Hungarian conventions (see e.g., [12], [13], [14]).

In this section, we show the modifications we have to introduce in order to make the new style files more complete.

Spacing around binary operators and relations Spacing is very important in mathematical expressions. (LA)TEX typeset mathematical formulae in a nice form, however, our standards slightly differ from those used in Hungarian typesetting.

In mathematical typesetting, one third normal spacing is used around binary operators and two thirds normal spacing around relations. (LA)TEX uses three mglue parameters called `\thinmuskip`, `\medmuskip` and `\thickmuskip`, for adjusting the spaces around elements of mathematical formulae. For setting these parameters to the required measures, we have given new values to these parameters:

```
\thickmuskip=4mu plus  2mu minus 4mu
\medmuskip=2mu plus 1.5mu minus 2mu
\thinmuskip=3mu
```

where `2mu` equals one third normal spacing. (The strange thing is that after the modification the thin space becomes wider than the medium space.) With the default values, the layout is

$$a + b - c/d * y \circ x = z, \tag{1}$$

and after the modifications

$$a+b-c/d*y\circ x=z. \tag{2}$$

Line break If TEX breaks a line after a binary operator or a relation in an inline mathematical formula, the sign has to be repeated on the next line.

Formulae cannot be broken at `\cdot` or slash.

Exclamation mark This sign has a special mathematical meaning, which forces us to handle it differently than the others.

For the most part, exclamation mark means the factorial sign in math mode. In this usage, it has to be followed by a small space without glue. Changing the class of this character from closing (number 5) to punctuation (number 6) (see e.g., [5], [11]), the problem has been solved in most cases. From the source code `{{k!n!(b-1)!} \choose {h!m!}} =1` we have

$$\binom{k!\,n!\,(b-1)!}{h!\,m!} = 1$$

which is an acceptable solution of our problem in most cases.

However, when a punctuation mark is followed by a binary operator, the spacing needs correction:

$$n!+k!+5a+6b,$$

its source code: `n!+k{!}+5a+6b`.

Space after commas In the Hungarian language, the decimal character is the comma, so we changed the default class of comma in math mode to 0, i.e., `\mathcode‘\,="013B` results in a decimal "point" in math mode, too:

$$F_i(x,y) = y^i + 1{,}3x \quad x,\ y \in A,\ i = 1,\ 2,\ 3, \ldots$$

This modification reduces the number of mistakes in the layout, although, we have to type approximately the same number of characters.

What next?

At the moment there is no standard way of writing text easily in Hungarian using plain TEX. The authors generally make and use their own macros. It is necessary to write style files, whose layout keeps

our typographic traditions and uses the modifications mentioned above.

The situation is better with LaTeX as babel offers an acceptable output and even some nice features. With some minor changes it can be improved enough to be closer to the needs of professional publishers: for example, changing the measure of white space produced by `\chapter`, `\section`, etc. commands, changing the dot or the spacing written after paragraph titles to a small typographic character, changing the appearance of footnotes to our traditions, and that of captions of tables and figures according to font sizes and types.

For (La)TeX, we would like to propose some additional modifications in math typesetting. We also plan to design Hungarian special ligatures (for the pairs *gy*, *gj*, *gz*), and even perhaps new fonts.

Last but not least we mention the different TeX-variants, like ε-TeX or Ω, the usage of which can help our work.

There is a lot to do in the future.

References

[1] Braams, Johannes. "Babel, a multilingual package for use with LaTeX's standard document classes." available from CTAN, `/macros/latex/required/babel`, 2001.

[2] Bardóczy, Irén. *Magasnyomó formakészítés.* (Making frames for letter-press printing), 3rd edn, Textbook, Budapest, Műszaki Könyvkiadó, 1979.

[3] Gyurgyák, János. *Szerkesztők és szerzők kézikönyve.* (Manual for Editors and Authors) Budapest, Osiris, 1998.

[4] Kehr, Roger. "xindy, A Flexible Indexing System." available from CTAN, `/indexing/xindy`, 1998.

[5] Knuth, Donald E. *The TeXbook.* Reading, MA, Addison-Wesley, 1988.

[6] Lamport, Leslie. "MakeIndex, An Index Processor For LaTeX." available from CTAN, `/indexing/makeindex`, 1987.

[7] Mayer, Gyula. "A TeX és LaTeX elválasztási modulja, 2002." (Hyphenation module for TeX and LaTeX) `http://www.typotex.hu/huhydok.pdf`

[8] Megyesi, Beáta. "The Hungarian Language, A Short Descriptive Grammar." `http://www.speech.kth.se/~bea/hungarian.pdf`

[9] Morphologic. "MSPELL." `http://www.morphologic.hu/en/en_mspell.htm`

[10] Németh, László. "Magyar ISPELL." `http://www.szofi.hu/gnu/magyarispell/`

[11] Salomon, David. *NTG's Advanced TeX Course: Insights & Hindsight.* Groningen: NTG, 1992.

[12] Szántó, Tibor. *Könyvnyomtatás – tipográfia.* (Printing – Typography), 2nd edn, Budapest, Műszaki Könyvkiadó, 1964.

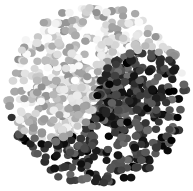
[13] Szántó, Tibor. *Könyvtervezés.* (Designing books) Budapest, Kossuth Nyomda, 1988.

[14] Timkó, György (ed.). *Helyesírási és tipográfiai tanácsadó.* (Orthographic and Typographic Guide) Budapest, Nyomdaipari Egyesülés, 1971.

[15] Virágvölgyi, Péter. *A tipográfia mestersége – számítógéppel.* (Craft of Typography – by computer) Budapest, Tölgyfa, 1998.

The Tao of Fonts

Włodzimierz Bzyl
matwb@univ.gda.pl



Abstract

Fonts are collections of symbols which allow people to express what they want to say, what they think or feel. Writing is a technique and as each technique has something to offer and has limitations. For example, the shapes of symbols depend on the tools and materials used for writing.

In the first part, I illustrate some writing techniques with examples. In the second part, I present a new technique for creating fonts and illustrate it with several examples. It is based on the METAPOST language [1] and could be viewed as Knuth's method [2–5] adapted to METAPOST. Knuth uses METAFONT to program fonts and an `mf` compiler to translate programs into bitmap fonts. Unfortunately, today's standards are based on PostScript fonts [6–9]. So, to keep the Knuth's idea of programmable shapes alive, fonts should be programmed in PostScript. This is difficult, because PostScript is a low level language.

The approach presented here is to program fonts in METAPOST. Although the `mpost` compiler is not able to output a font directly, its output can be assembled into a PostScript font [10–13]. A font programming environment is based on a revised version of the `mft` pretty-printer. The original `mft` understands only METAFONT, but changed `mft` understands METAPOST too.

In the third part, I present a simple font programmed as a Type 3 and as a Type 1 font. These examples will give an idea of font programming.

In the Appendix, I present a detailed description of Type 3 fonts [6, 9].

Typographical journey

Exploring type [with computer] is fun, and ultimately, it changes the way you think about type and work with it.

— ROB CARTER, Experimental Typography

Throughout history people used have symbols to visually encode thoughts and feelings. The oldest example I was able to find in the literature is an inscription from La Pasiega cave.

Fig. 1. Inscription from La Pasiega cave (Spain, ca. 10000 B.C.)

Unfortunately, the meaning of the symbols was lost in the past, so that we don't know how to decode this inscription. As a result, we don't know exactly what it means — we can only guess. Other

Fig. 2. 'Magical' stamps

ancient symbols are found on pikes and bows. It is assumed that they are some kind of owner's signature or that they have a magical value, so that they bring luck to the owner, etc. Nowadays, we still use symbols in similar way. Probably everyone has at least once received a mail overprinted with "CONFIDENTIAL – you have been chosen to be rich. You can take part in our lottery draw. All you have to do is to subscribe our magazine."

Pictographs, ideograms, and alphabets have been written and reproduced on papyrus, stones, wood, clay tablets, paper, and computer displays;

and different techniques have been used to write, carve, cut, and print symbols.

Handwriting used to be the most common technique. The Phoenicians invented an alphabetic font which is a precursor of the Greek, Latin, and Cyrillic fonts. The inscription of Fig. 3 was originally written on papyrus.

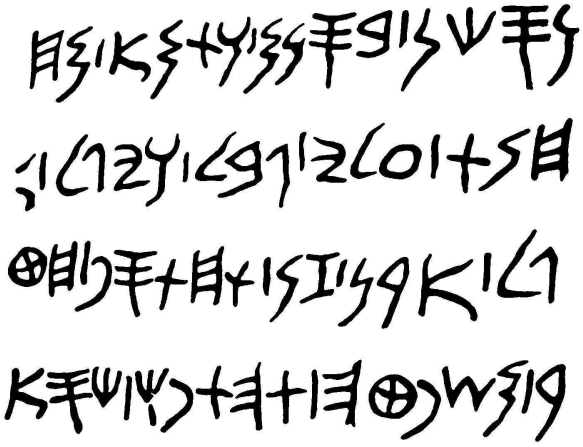


Fig. 3. Phoenician inscription
(Byblos, 1100 B.C.)

Handwriting can reveal the author's personality, which makes this technique interesting. Wouldn't it be nice to have a psychological portrait of this author?

Fig. 4 shows a fragment of a poem written by probably the greatest Polish poet Cyprian Kamil Norwid. This poem seems to move every Pole who reads it. I think that the same poem printed along with several others in Computer Modern, Garamond, Times, or Palatino would not have the same impact on readers. Maybe this is why I don't like reading poems which all look the same.†

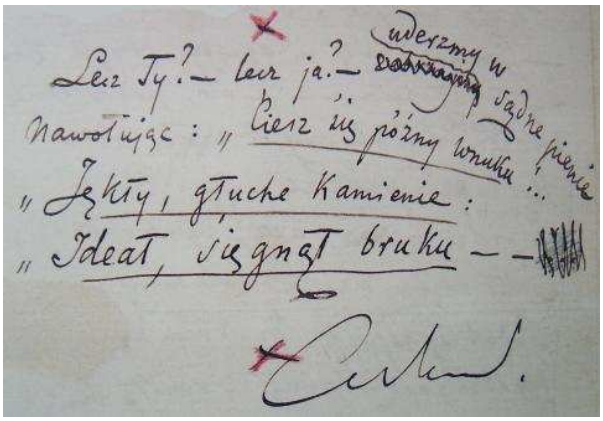


Fig. 4. Cyprian K. Norwid, *Vade-mecum*
(manuscript in Polish, 1865–1866)

Fig. 5. *Calendarium Parisiense*
(manuscript Latin, ca. 1425)

But the main problem caused by handwriting (and computer typesetting too) is the appearance of overfull and underfull lines. These, in some cases, can not be eliminated. Fig. 5 shows the earliest example I was able to find.

Another technique is cutting in stone. The shapes in Fig. 6 are more regular, partly because bigger letters were sketched beforehand.

This picture shows the earliest example of serifs ever found. The serifs are functioning here as a way of finishing letters, which otherwise would be irregular and wiggly ended. Nowadays, we think that serif fonts are easier and quicker to read. This is generally true, because letters without serifs in some cases looks similar to each other, for example: I – l – 1.*

† But I would be very grateful if my doctor chose to use Computer Modern on my prescriptions.

* I cannot imagine books or newspapers carved in stone. Nevertheless, there was once an exception: I have seen Fred Flintstone reading newspapers. But this was a long time ago down in the Bed Rock.

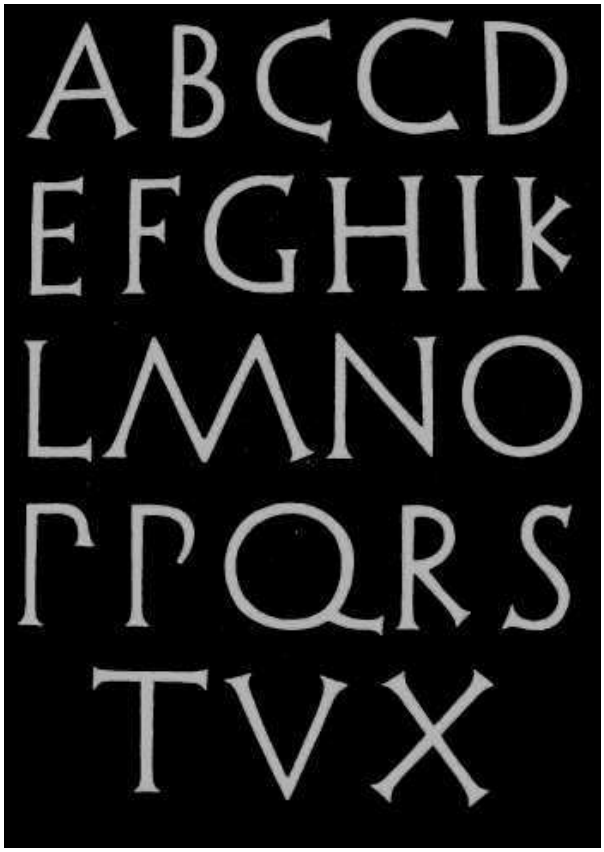


Fig. 6. Rome, (ca. 200 B.C.)

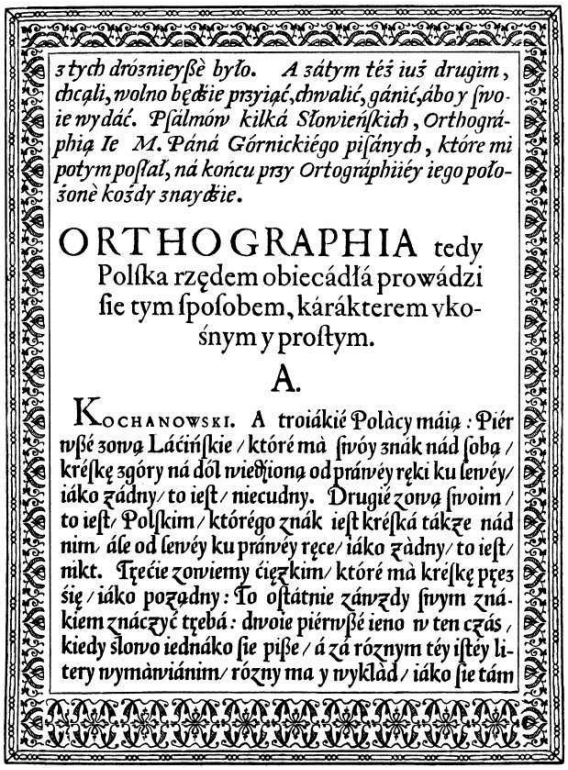


Fig. 7. *New Polish Character*
(Jan Januszowski, 1594 Cracow)

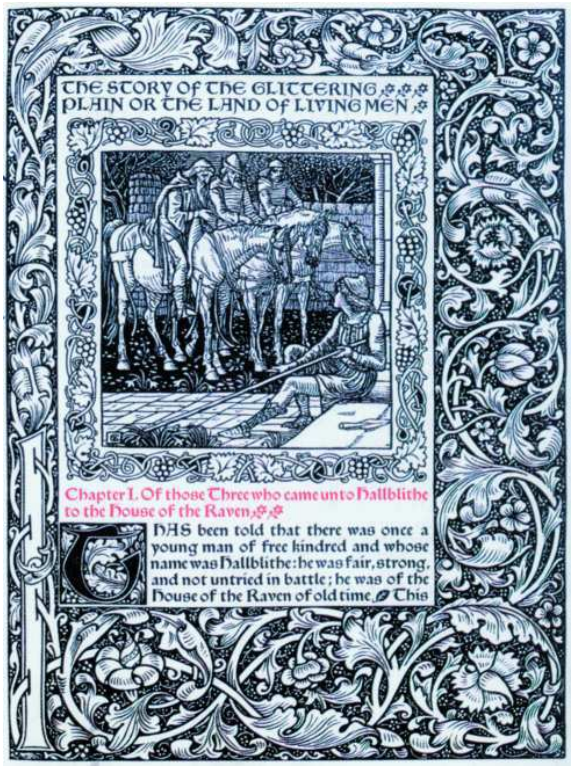


Fig. 8. *The Story of the Glittering Plain*
(Kelmscott Press, 1891)

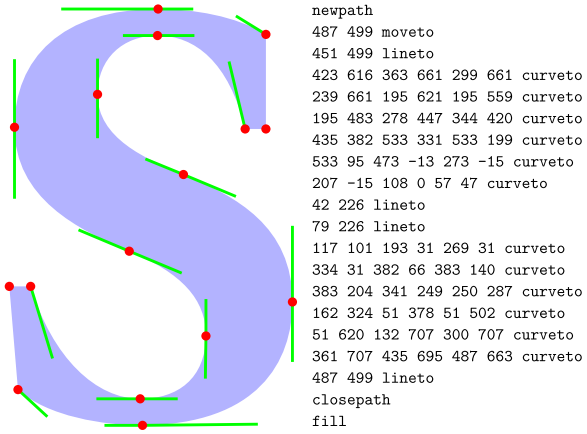
Fig. 7 shows the 32nd page of one of the first printed books in Poland. This book is very important, because it contains designs for Polish diacritics: aogonek, eogonek etc. The borders were cut in wood and letters in metal. The printed letters are much smaller than carved ones, so cutting them was a real challenge, yet even tiny serifs are present. This technique was perfected over the years. The results are seen on Fig. 8. Here borders were cut in wood and the type was cut in metal.

Finally we get to Fig. 9 showing what computers are good at. The 'shapes', drawn by a computer, are almost ideal. Typographical embellishments are not present — instead the letters are colored and a background photo is used to improve the typography of the page.

Each new technique starts from the point where the old one ends.

When we make a letter with a computer, the mouse is used to put points on an 'imaginary sheet'. As the points are laid down, the computer joins them with curves. Next, the inflexion points (red) and the endings of tangents (green lines) are adjusted by hand. At the same time, the computer

Fig. 9. "Playboy" (Polish edition, 2001)



```
newpath
487 499 moveto
451 499 lineto
423 616 363 661 299 661 curveto
239 661 195 621 195 559 curveto
195 483 278 447 344 420 curveto
435 382 533 331 533 199 curveto
533 95 473 -13 273 -15 curveto
207 -15 108 0 57 47 curveto
42 226 lineto
79 226 lineto
117 101 193 31 269 31 curveto
334 31 382 66 383 140 curveto
383 204 341 249 250 287 curveto
162 324 51 378 51 502 curveto
51 620 132 707 300 707 curveto
361 707 435 695 487 663 curveto
487 499 lineto
closepath
fill
```

Fig. 10

redraws the shape (see Fig. 10). We have replaced a chisel by a mouse and wood/metal by the computer screen. With these tools, we can polish the shape as long as we wish, and we can not ruin the shape with the one wrong decision as can happen with traditional tools and materials.

One of the limitations of this technique is shown in Fig. 11. Printing a symbol by computer means putting the imaginary sheet on the screen. This sheet can be expanded, contracted, or skewed — we can apply any geometrical transformation to it. Unfortunately, the results may not be satisfactory;

Fig. 11

for example, the stop sign in the third row has lost its white border and the inscription is hardly readable. We could repair this if it would be possible to drop the border and scale the inscription less, but the operation is unfeasible, because the computer does not know which numbers in the character design are responsible for the border and which for the inscription.

So the only way to produce a better font at small sizes is to make it from scratch. Since some will inevitably use an enlarged version of this font instead of the original one, and chaos will ensue. Imagine a country where each town has slightly different traffic signs!

Programming fonts

Typographic standards make type more readable, but readability has become a relative concept. The immediacy of personal computers and the World Wide Web has raised the level of 'typographic literacy': computers are used to stretch typographic boundaries [14].

Before we start to explore type with computers, we should ask: *What is the right way to create digitized patterns for printing or displaying?* In this section, I will try to convey some of my excitement about experiments with the tools I have created, since I think that I am going in the right direction [see also 3].

To play with and to explore fonts I use a set of UNIX tools. To this set I added the DOS batch files forming the METATYPE1 package by the JNS team [11], which I converted to UNIX scripts.

The language for font programming is METAPOST. To make a METAPOST font usable we must convert it into something that printing and typesetting systems can understand. I chose PostScript Type 1 or Type 3 font programs, mainly because Ghostscript — a free PostScript interpreter — is available on almost every computer platform.

The Linux version of the tools consists of METAPOST font libraries and four scripts:

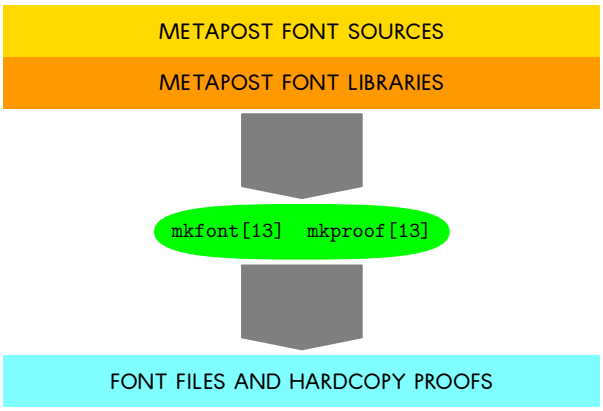


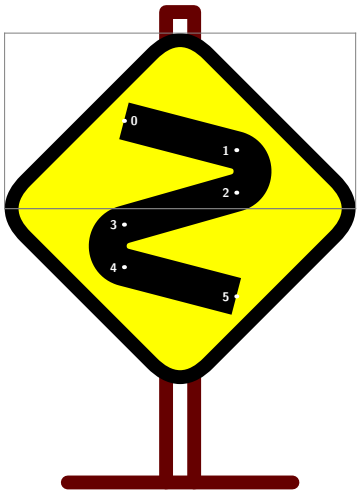
Fig. 12. Fonts programming

`mkfont1` – converts METAPOST font sources to Type 1 font: a shell script that uses programs `mpost`, `t1asm` from the t1utils package and `awk`.

`mkfont3` – converts METAPOST font sources to Type 3 font; a perl script that calls `mpost`

`mkproof1` and `mkproof3` – scripts that produce hardcopy proofs and are used as debugging tools: they call `mpost` and the `mft` pretty-printer.

SIGN-000.MP




```
beginpic(127, 250, 125, 0); "Dangerous bend";
    draw post; draw info_signboard;
    clearxy;
    % the dangerous bend
    numeric heavyline; heavyline := 27;
    x5 = w − x0; x5 − x0 = 80; x1 = x2 = x5; x0 = x3 = x4;
    y0 = −y5 = 1/2 h; y1 = −y4 = 1/3 h; y2 = −y3 = 1/11 h;
    pickup pencircle scaled heavyline;
    interim linecap := butt;
    draw z0 -- z1{z1 − z0} .. z2 --- z3 .. z4{z5 − z4} -- z5
        withcolor c.Dangerous Bend;
    labelcolor := white; dotcolor := white;
    labels lft(1, 2, 3, 4, 5); labels rt(0);
endpic;
```


Fig. 13. Hardcopy proof of the dangerous bend

In font programming two type of errors appear: bugs in font program and design errors. Bugs are treated in an ordinary way. To catch design errors I use hardcopy produced by the `mft` program.

We have the tools, so it's high time to start programming. Let's start from the beginning.

The Phoenician font lacks vowels. There are three ways of writing with this font. Lines may be written from left to right, right to left, or left to right, right to left, etc., with letters on every other line reflected vertically. It could be a real challenge to typeset a Phoenician script with TEX.

To [our] Lady Ishtar
This is the holy place

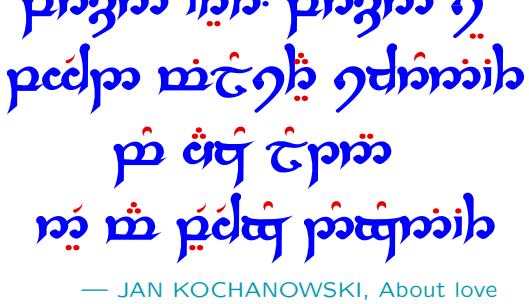
Fig. 14. Phoenician font [26]

TEX could be used to communicate with the STAR TREK crew: we only need their font. No problem: there is nothing special about this font, except the extraordinary shapes of the symbols and the use of few ligatures.

Listen sons of Kahless!
Listen his daughters!

Fig. 15. Klingon font [23]

We can also send our classic love poems to elves. The elves write vowels over the preceding letter, unless it is also a vowel. This case, and the case when a vowel starts a word, are handled by other rules [18].



— JAN KOCHANOWSKI, About love

Fig. 16. Tengwar font [24]

It appears that that all these rules can be implemented with an appropriate ligature and kerning table.

What was considered unreadable yesterday is readable today. People are more visually sophisticated and typographically savvy than ever before, so my next font contains ideograms for love and for some other emotions, as well as letters.

Fig. 17. Redis font (see also [28])

Special math fonts can be useful, too. We could use them on title pages or on slides. In the example below, math characters are colored according to their math class as defined in plain format. Below, binary operators are painted in green, large operators in magenta, etc. (In the pdf version, not on paper.)

$$M = \begin{array}{c} \\ C \\ I \\ C' \end{array} \begin{array}{c} \begin{array}{ccc} C & I & C' \end{array} \\ \begin{pmatrix} 1 & 0 & 0 \\ \beta & 1-\beta & 0 \\ 0 & \xi & 1-\xi \end{pmatrix} \end{array}$$

$$\begin{aligned} \left(\int_{-\infty}^{\infty} e^{-x^2}\,dx\right)^2 &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-(x^2+y^2)}\,dx\,dy \\ &= \int_0^{2\pi}\int_0^{\infty} e^{-r^2} r\,dr\,d\theta \\ &= \int_0^{2\pi} \left(-\frac{e^{-r^2}}{2}\Big|_{r=0}^{r=\infty}\right) d\theta \\ &= \pi \end{aligned} \tag{78}$$

Fig. 18. New Punk Math font (see also [19])

Fig. 19 shows a piece of text typeset with a computer replica of the font used in the Polish Alphabet Primer by Falski. I learned to write letters with the help of Falski's primer, as did my wife and my daughter. In fact, all children in Poland learn Falski's letters.

DIALOGS ARE REWRITTEN AGAIN
TO SPARE THE AUTHOR AND THE
OTHER CHARACTERS THE SHAME OF
SOUNDING AS INARTICULATE AS
THEY INVARIABLY DO OR WOULD IF
THEIR SENTENCES ALMOST INVARIABLY
BEGUN WITH THE WORD DUDE
AS IN FOR EXAMPLE DUDE SHE
DIED WERE MERELY TRANSCRIBED

Fig. 19. Ala font [27]

The characteristic features of this font are listed below:

- size of characters: BIG,
- width: PROPORTIONAL,
- slanting: UPRIGHT,
- interletter spacing: BIG,
- uniformity of pressure: CONSTANT,
- strength of pressure: AVERAGE,
- interword spacing: BIG,
- overall appearance: OVAL,
- readability: CLEAR.

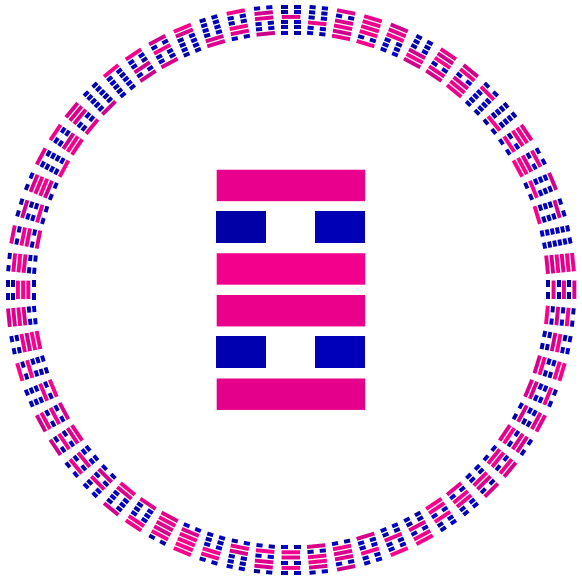
Handwriting can reveal the personality of the writer. The writer in this case is my computer so we can get 'his' psychological portrait easily: *A person which writes this way is well-wishing, Easy to cooperate with and friendly. Usage of uppercase letters and big interletter spacing indicates this. Constant pressure means emotional maturity. Oval appearance might mean uncertainty and submissiveness. Finally, constant and average pressure and wide characters indicate an uneasy and over excitable person.*

The most important thing about this example is how easy it is to make this font to look differently, for example more 'technical'. It suffices to change few numbers which define this font. But it would be difficult to simulate other important features of handwritten scripts, such as variable baseline, variable letters shape, pressure of script.

The next example shows the I-Ching font (see also [29]). The I Ching or "The Book of Changes" is an old Chinese oracle. This font could be used to do divinations. With computers it is easy. Ask a question, press Enter key and your computer will do the rest. On the next page I put results obtained

LI

To Shine Brightly, to Part



INTERPRETATION

To Part. It is useful to stand firm and behave well. This will bring success. Take care of the cows. There will be good fortune.

Interpretation of the 4th change line

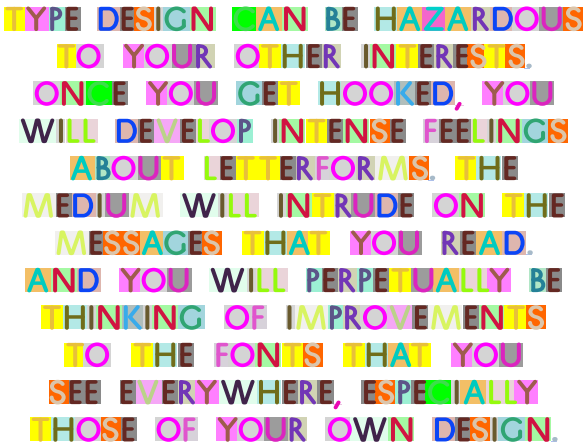
It comes unexpectedly. It is like a fire which dies down and is discarded.

during my presentation at the TUG 2002 meeting in Trivandrum.

Nowadays, children in Poland have more and more problems with orthography. There are many pairs of letters which cause them problems. For example the letters 'oacute' and 'u' are pronounced the same way, 'b' and 'p' are pronounced almost the same way, etc. The following trick is used to teach orthography to children with dyslexia [22]. Each problematic letter is mapped to a crayon with different colors corresponding to different letters. Instead of writing a problematic letter, the child uses the appropriate crayon to draw a rectangle. After a while the crayons are removed. This method is supplemented by appropriate books and dictionaries.

The following 'text' demonstrates the extreme case in which every letter is problematic.

I like this kind of writing, so why take off crayons?



— DONALD E. KNUTH, The METAFONTbook

Type 3 font example

Fonts are collections of programmed shapes. There are several kinds of fonts. Each type of font has its own convention for organizing and representing the information within it. The PostScript language defines the following types of fonts [8, p. 322]: 0, 1, 2, 3, 9, 10, 11, 14, 32, 42. Text fonts are mostly of Type 1, which are programmed with special procedures. To execute efficiently and to produce more legible output, these procedures use features common to collections of black & white letter-like shapes. The procedures may not be used outside a Type 1 font. While any graphics symbol may be programmed as a character in a Type 1 font, non-letter shaped symbols are better served by the Type 3 font program which defines shapes with ordinary PostScript procedures including those which produce color. Other font types are used infrequently.

Although Type 3 fonts are PostScript programs, I prefer to program shapes in the METAPOST language and convert the `mpost` output into a Type 3 font, because METAPOST simplifies the programming due to its declarative nature. In PostScript each curve is built from lines, arcs of circle and Beziér curves [p. 393, 9]. For complicated shapes this requires a lot of nontrivial programming. METAPOST implements a 'magic recipe' [10] for joining points in a pleasing way, which helps a lot. Even if you are not satisfied with the shape, you can give the program various hints about what you have in mind, therefore improving upon the automatically generated curve. To use a font with TEX the font metric file is required. It contains data about width, height and depth of each shape from the font. Because `mpost` could generate metric file on demand, fonts prepared with METAPOST are immediately usable with TEX.

Creation of a Type 3 font is a multi-step process.

1. A font must be imagined and designed.
2. The design must be programmed. This step is supported by a specially created library.
3. The METAPOST font program must be compiled.
4. The files thus created must be assembled into a font. This task is done by the `mkfont3` Perl script.

Additionally, the font must be made available to TEX and instructions must be given to tell TEX how to switch to this font.

Let's create a font which contain one character named plus. Use an ascii text editor — it does not have to be your favorite, any such editor works — to create a file called `plus-000.mp` that contains the following lines of text.

Each font program should name the font it creates.

```
font_name "Plus-000";
```

These names are merely comments which help to understand large collections of PostScript fonts.

```
family_name "Plus";
font_version "0.0final";
is_fixed_pitch true;
```

and following names play similar rôle in the TEX world.

```
font_identifier:="PLUS 000";
font_coding_scheme:="FONT SPECIFIC";
```

The `mpost` program does all its drawing on its internal 'graph paper'. We establish a 100×100 coordinate space on it.

```
grid_size:=100;
```

The font matrix array is used to map all glyphs to PostScript's 1×1 coordinate space. This convention allows consistent scaling of characters which come from different fonts.

```
font_matrix
  (1/grid_size,0,0,1/grid_size,0,0);
```

This particular font matrix will scale a plus shape by the factor 1/100 in the x dimensions and by the same factor in the y dimension. If we had choosen scaling by the factor 1/50 then the plus shape would have appeared twice as large as characters from other fonts.

The data below provides information about how to typeset with this font. A font quad is the unit of measure that a TEX user calls one '`em`' when this font is selected. The normal space, stretch, and shrink parameters define the interword spacing when text is being typeset in this font. A font like this is hardly ever used to typeset anything apart from the plus, but the spacing parameters have been included just in case somebody wants to typeset several pluses separated by quads or spaces.

```
font_quad:=100;
font_normal_space:=33;
font_normal_stretch:=17;
font_normal_shrink:=11;
```

Another, more or less ad hoc, unit of measure is `x_height`. In TEX this unit is available under the name '`ex`'. It it used for vertical measurements that depend on the current font, for example for accent positioning.

```
font_x_height:=100;
```

The plus font is an example of a parameterized font. A single program like this could be used to produce infinite variations of one design. For example, by changing the parameters below we could make the plus character paint in a different color, or make it thicker.

```
color plus_color;
plus_color:=red;
u:=1; % unit width
pen_width:=10;
```

The `mode_setup` macro could be used to override all the settings done above. Typically, it is used to tell the `mpost` program to generate a font metric file or proofsheets. Additionally, `mode_setup` could execute any piece of valid METAPOST code at this point. For example, we could change the color of plus to yellow and the pen width to 5 units. The code to be executed could be read from a separate file (see below on how to prepare and use such a

file). Thus we can make a variation of this design or *re-parameterize* the font without changing the master `plus-000.mp` file. Such a mechanism is required, to avoid populating our hard disks with similar files.

```
mode_setup;
```

The Type3 library makes it convenient to define glyphs by starting each one with:

beginpic (⟨code⟩, ⟨width⟩, ⟨height⟩, ⟨depth⟩)

where ⟨code⟩ is either a quoted single character like `"+"` or a number that represents the glyph position in the font. The other three numbers say how the big the glyph bounding box is. The command **endpic** finishes the plus glyph.

Each **beginpic** operation assigns values to special variables called `w`, `h`, and `d`, which represent respective width, height, and depth of the current glyph bounding box. Other pseudo-words are part of METAPOST language and are explained in [6].

```
beginpic("+",100u,100,0); "+  plus";
  interim linecap:=butt;
  drawoptions(withcolor stem_color);
  pickup pencircle scaled stem_width;
  draw (w/2,-d)--(w/2,h);
  draw (0,(h-d)/2)--(w,(h-d)/2);
endpic;
```

Finally, each font program should end with the **endfont** command.

```
endfont
```

Now, we are ready to compile the font with `mpost` and assemble generated glyphs into Type 3 font with one command:

```
mkfont3 plus-000
```

To use `Plus-000` font in a TEX document it suffices to insert these lines:*

```
\font\X=plus-000 at 10pt
\centerline{\X +\quad+ +++ +\quad+}
```

This code produces the seven red pluses below.

A font cannot be proved faultless. If some glyphs are defective, the best way to correct them is to look at a big hardcopy proof that shows what went wrong. The hardcopy for the `Plus-000` font could be generated with the following shell command:

```
mkproof3 -u plus-000.map plus-000.mp
```

* To see characters from a PostScript `Plus-000` font, the DVI file must be processed by DVIPS (see the explanations at the end of this section).

As mentioned above, it is not wise to make one-time-only variation of a font by changing the font source. To change font parameters `mode_setup` is used in conjuction with the `change_mode` macro. used. I will explain this last sentence with an example.

Assume that fictitious document `doc.tex` uses `Plus-000` font and the font program reside in the file `plus-000.mp`.

The default color of the plus symbol is red. To create a variation of the font with the plus symbol painted in yellow we re-parameterize it using a file named `doc.mp`, with the following content:

```
mode_def plus_yellow = message "yellow +";
  final_; % create metric file
  font_name "Plus-b00";
  plus_color:=(1,1,0);
enddef;
```

Now, we can create a TFM file, Type 3 font, and dvips fontmap file with the command:

```
mkfont3 --change-mode=doc,plus_yellow \
  --change-name=plus-b00 plus-000.mp
```

To test the font, create a file named `doc.tex` with the following content:

```
\font\Y=plus-b00 at 10pt
\centerline{\Y +\quad+ +++ +\quad+}
```

typeset it and convert to PostScript:

```
tex doc.tex
dvips -u plus-b00.map doc.dvi -o doc.ps
```

This should generate file named `doc.ps` which may be viewed and printed, for example with the Ghostscript program. The programmed yellow plus is printed below.

Generating hardcopy proofs, compiling fonts, typesetting documents requires remembering and executing a lot of shell commands. Here, the `make` utility helps a lot [20].

Type 1 font example

Type 1 font programming differs from Type 3 font programming. Type 3 glyphs can use any PostScript command, but Type 1 glyphs use a subset of PostScript. Moreover, we must construct an outline of glyph instead of drawing it. The outline is filled when the glyph is printed.

Each METAPOST font should input the METAPOST Type1 library. The library contains macros which help to compute outlines, and to output

various font data to several files. These data are used by the `mkfont1` script which assembles Type 1 font and `mkproof1` script which typesets hardcopy proofs.

The Type 1 font programmed below contains nothing but a plus symbol. Let's start with reading basic macros.

```
input type1;
```

Next follows the usual font administration stuff. Each font should define several variables [9, Tables 5.1–4].

```
pf_info_familyname "Plus";
pf_info_fontname "Plus-Regular";
pf_info_weight "Normal";
pf_info_version "1.0";
pf_info_fixedpitch true;
pf_info_author "Anonymous 2002";
pf_info_creationdate;
```

The `mpost` program does all its drawing on its internal 'graph paper' with 1000×1000 coordinate space on it. The data below provides information about how to typeset with this font.

```
pf_info_quad 760;
pf_info_capheight 760;
pf_info_xheight 760;
pf_info_space 333;
```

The `adl` suffix here is a mnemonic for *Ascender*, *Descender*, and *Lineskip*.

```
pf_info_adl 750, 250, 0;
```

The PostScript **fill** operator is used to paint the entire region closed by the current path. For each path, the *non-zero winding number rule* [9, p. 161] determines whether a given point is inside a path. This behaviour is simulated by the `Fill` and `unFill` macros. The `fill_outline` macro, for each closed path stored in the array `s[1..s.num]`, fills or unfills it based on its *turning number* [4, p. 111].

```
def fill_outline suffix s =
  for i:=1 upto s.num:
    if turningnumber s[i] > 0: Fill
    else: unFill fi s[i];
  endfor
enddef;
```

The plus sign has squared-off ends. Macro `butt_end` simplifies the task of cutting of ends of paths.

```
def butt_end(text nodes) =
  cut(rel 90)(nodes)
enddef;
```

A horizontal line of the same width as a vertical line seems thicker. To avoid this optical illusion we use an elliptical pen.

```
numeric px; px:=100;
numeric py; py:=90;
default_nib:=fix_nib(px,py,0);
```

These names are intended to make the code more readable.

```
path vertical_stem, horizontal_stem;
path glyph;
```

Each glyph should be defined within a block defined by `beginfont` and `endfont` commands.

```
beginfont
```

Programmed symbols must be given names as well as positions in the font.

```
encode("plus",43);
```

Each glyph starts with **beginglyph** and ends with **endglyph** macro. The following macros initialize several variables, used for the glyph data bookkeeping.

```
standard_introduce("plus");
beginglyph("plus");
```

For convenience, the width, height and depth of the character are assigned to variables *w*, *h*, and *d*.

```
w:=760; h:=760; d:=0;
```

The horizontal and vertical bars of the plus glyph are centered with respect its bounding box.

```
z0=(w/2,d); z1=(w/2,h);
z2=(0,(h-d)/2); z3=(w,(h-d)/2);
```

To draw paths `z0--z1` and `z2--z3` the pen with a `default_nib`-shaped nib is used. The macro `pen_stroke` finds the outline of each path. Outlines are assigned to the paths `vertical_stem` and `horizontal_stem`. The macro `butt_end` cuts off the ends of these paths at times 0 (beginning) and 1 (end).

```
pen_stroke(butt_end(0,1))(z0--z1)
  (vertical_stem);
pen_stroke(butt_end(0,1))(z2--z3)
  (horizontal_stem);
```

Programming a Type 1 glyph means constructing its outline (which could be made up of several cyclic paths). The macro below finds the outline of the paths constructed above and stores it in the array named in the second argument.

```
find_outlines
  (vertical_stem,horizontal_stem)(glyph);
```

Now, we are ready to draw the plus symbol.

```
fill_outline glyph;
```

Finally, we fix the width of the glyph to `w` and its left and right sidebearings to 0.

```
fix_hsbw(w,0,0);
```

Each symbol should include so-called *hints* [8, p. 56–57] that make it render better on a wide variety of devices.

```
fix_hstem(py)(horizontal_stem);
fix_vstem(px)(vertical_stem);
```

To make our hardcopy proofs more readable we define some construction points (see the figure below).

```
dotlabels(0,1,2,3);
```

The last two macros end the subprogram for plus symbol and the whole font program.

```
endglyph;
endfont;
```

Now, we can create a TFM file, Type 1 font, and dvips fontmap file with the command:

```
mkfont1 plus.mp
```

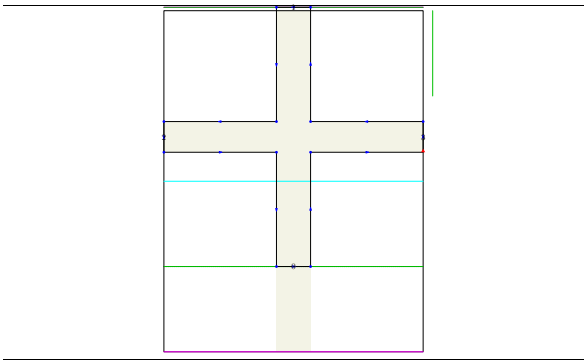
The plus character just constructed is used to print the divider line below.

+ +++++ +

The hardcopy proof below was typeset with the command:

```
mkproof1 -u plus.map plus.mp
```

```
def fill_outline suffix s =
    for i := 1 upto s_num:
      if turningnumber s[i] > 0: Fill else: unFill fi s[i] ;
    endfor
enddef ;
def butt_end(text nodes) = cut(rel 90)(nodes) enddef ;
numeric px ; px := 100 ;
numeric py ; py := 90 ;
default_nib := fix_nib(px, py, 0) ;
path vertical_stem, horizontal_stem, glyph ;
beginfont
```



```
encode("plus", 43) ;
standard_introduce("plus") ;
beginglyph("plus") ;
    w := 760 ; h := 760 ; d := 0 ;
    z0 = (w/2, d) ; z1 = (w/2, h) ;
    z2 = (0, (h − d)/2) ; z3 = (w, (h − d)/2) ;
    pen_stroke(butt_end(0, 1))(z0 -- z1)(vertical_stem) ;
    pen_stroke(butt_end(0, 1))(z2 -- z3)(horizontal_stem) ;
    find_outlines(vertical_stem, horizontal_stem)(glyph) ;
    fill_outline glyph ;
    fix_hsbw(w, 0, 0) ;
    fix_hstem(py)(horizontal_stem) ;
    fix_vstem(px)(vertical_stem) ;
    dotlabels(0, 1, 2, 3) ;
endglyph ;
endfont ;
```

21:05 17 X 2002 PLUS 3

Appendix

This description is somewhat simplified in respect to the examples to be found in [6, 9].

Each Type 3 font should begin with two lines of comments.

```
%!PS-AdobeFont-1.0: Square 1.00
%%CreationDate: 1 May 2001
```

A Type 3 font consists of a single dictionary, possibly containing other dictionaries, with certain required entries. The dictionary of size 99 should suffice for fonts which consists of several characters.

```
99 dict begin
```

This dictionary should include following entries:

- Variable `FontType` indicates how the character information is organized; for Type 3 fonts it has to be set 3.
- Variable `LanguageLevel` is set to the minimum PostScript language level required for correct behavior of the font.
- Array `FontMatrix` transforms the character coordinate system into the user coordinate system. This matrix maps font characters to one-unit coordinate space, which enables the PostScript interpreter to scale font characters properly. This font uses a 1000-unit grid.
- Array (of four numbers) `FontBBox` gives lower-left (l_x, l_y) and upper-right (u_x, u_y) coordinates of the smallest rectangle enclosing the shape that would result if all characters of the font were placed with their origins coincident, and then painted. This information is used in making decisions about character caching and clipping. If all four values are zero, no assumptions about character bounding box are made.

```
/FontType 3 def
/LanguageLevel 2 def
/FontMatrix [ 0.001 0 0 0.001 0 0 ] def
/FontBBox [ 0 0 1000 1000] def
```

The `FontInfo` dictionary is optional. All information stored there is entirely for the benefit of PostScript language programs using the font, or for documentation.

- `FamilyName` — a human readable name for a group of fonts. All fonts that are members of such a group should have exactly the same FamilyName.
- `FullName` — unique, human readable name for an individual font. Should be the same name as one used when registering the font with the `definefont` operator below.
- `Notice` — copyright, if applicable.

- `Weight` — name for the "boldness" attribute of a font.
- `version` — version number of the font program.
- `ItalicAngle` — angle in degrees counterclockwise from the vertical of the dominant vertical strokes of the font.
- `isFixedPitch` — if true, indicates that the font is a monospaced font; otherwise set false.
- `UnderlinePosition` — recommended distance from the baseline for positioning underlining strokes (y coordinate).
- `UnderlineThickness` — recommended stroke width for underlining, in units of the character coordinate system.

```
/FontInfo <<
   /FamilyName (Geometric)
   /FullName (Square)
   /Notice (Type 3 Repository.
     Copyright \(C\) 2001 Anonymous.
     All Rights Reserved.)
   /Weight (Medium)
   /version (1.0)
   /ItalicAngle 0
   /isFixedPitch true
   /UnderlinePosition 0.0
   /UnderlineThickness 1.0
>> def
```

The `Encoding` array maps character codes (integers) to character names. All unused positions in the encoding vector must be filled with the name `.notdef`. It is special in only one regard: if some encoding maps to a character name that does not exist in the font, `.notdef` is substituted. The effect produced by executing `.notdef` character is at the discretion of the font designer, but most often it is the same as space.

```
/Encoding 256 array def
0 1 255
  {Encoding exch /.notdef put}
for
```

The `CharacterProcedures` dictionary contains individual character definitions. The name is not special — any name could be used — but this name is assumed by the `BuildGlyph` procedure below.

```
/CharacterProcedures 256 dict def
```

Each character must invoke one of the `setcachedevice` and `setcharwidth` operators before executing graphics operators to define and paint the character. The `setcachedevice` operator stores the bitmapped image of the character in the font cache. However, caching will not work if color or gray is used. In such cases the `setcharwidth` operator should be used, which is similar to `setcachedevice`, but declares that the character being defined is not to be placed in the font cache.

$w_x\ w_y\ l_x\ l_y\ u_x\ u_y$ `setcachedevice` –
w_x, w_y — comprise the basic width vector, i.e., the normal position of the origin of the next character relative to origin of this one
l_x, l_y, u_x, u_y — are the coordinates of this character bounding box

$w_x\ w_y$ `setcharwidth` –
$w_x\ w_y$ — comprise the basic width vector of this character

```
CharacterProcedures /.notdef {
   1000 0 0 0 1000 1000 setcachedevice
   1000 0 moveto
} put
Encoding 32 /space put
CharacterProcedures /space {
   1000 0 0 0 1000 1000 setcachedevice
   1000 0 moveto
} put
Encoding 83 /square put % ASCII 'S'
CharacterProcedures /square {
   1000 0 setcharwidth
   0 1 1 0 setcmykcolor % red
   0 0 1000 1000 rectfill
} put
```

The `BuildGlyph` procedure is called within the confines of a `gsave` and a `grestore`, so any changes `BuildGlyph` makes to the graphics state do not persist after it finishes.

`BuildGlyph` should describe the character in terms of absolute coordinates in the character coordinate system, placing the character origin at $(0, 0)$ in this space.

The Current Transformation Matrix (CTM) and the graphics state are inherited from the environment. To ensure predictable results despite font caching, `BuildGlyph` must initialize any graphics state parameter on which it depends. In particular, if `BuildGlyph` executes the `stroke` operator, it should explicitly set: dash parameters, line cap, line join, line width. These initializations are unnecessary if characters are not cached, for example if the `setcachedevice` operator is not used.

When a PostScript language interpreter tries to show a character from a font, and the character is not already present in the font cache it pushes *current font dictionary* and *character name* onto the operand stack. The `BuildGlyph` procedure must remove these two objects from the operand

stack and use this information to render the requested character. This typically involves finding the character procedure and executing it.

```
/BuildGlyph { % stack: font charname
  exch
  begin
  % initialize graphics state parameters
  % turn dashing off: solid lines
    [ ] 0 setdash
  % projecting square cap
    2 setlinecap
  % miter join
    0 setlinejoin
  % thickness of lines rendered by
  % execution of the stroke operator
    50 setlinewidth
  % the miter limit controls the stroke
  % operator's treatment of corners;
  % this is the default value and it
  % causes cuts off mitters at
  % angles less than 11 degrees
    10 setmiterlimit
    CharacterProcedures exch get exec
  end
} bind def
currentdict
end % of font dictionary
```

Finally, we register the font name as a font dictionary defined above and associate it with the key `Square`. Additionally the `definefont` operator checks if the font dictionary is a well-formed.

```
/Square exch definefont pop
```

If the following lines are not commented out the Ghostscript program (a public domain PostScript interpreter) will show the text below online. Obviously, these lines should be commented out in the final version of the font program.

```
/Square findfont
  72 scalefont setfont
0 72 moveto (S) show
showpage
```

References

[1] John D. Hobby. 1992. *A User's Manual for MetaPost*. Technical Report 162. AT&T Bell Laboratories, Murray Hill / New Jersey. Available online as a part of METAPOST distribution.

[2] Donald E. Knuth. 1982. "The Concept of a Meta-Font." *Visible Language* **16**, 3–27.

[3] Donald E. Knuth. 1985. "Lessons Learned from METAFONT." *Visible Language* **19**, 35–53.

[4] Donald E. Knuth. 1986. *The METAFONTbook*. American Mathematical Society and Addison Wesley.

[5] Donald E. Knuth. 1992. *Computer Modern Typefaces*. Addison Wesley.

[6] Adobe Systems Incorporated. 1985. *Tutorial and Cookbook*. Addison Wesley.

[7] Adobe Systems Incorporated. 1992. *The PostScript Font Handbook*. Addison Wesley.

[8] Adobe Systems Incorporated. 1993 (3rd printing), Version 1.1. *Adobe Type 1 Font Format*. Addison Wesley.

[9] Adobe Systems Incorporated. 1999 (3rd printing). *PostScript Language Reference Manual*. Addison Wesley.

[10] Bogusław Jackowski et al. 1999. "Antykwa Półtawskiego: a parameterized outline font." EuroTEX 99 Proceedings. Ruprecht-Karls-Univerität Heidelberg, 117–141.

[11] Bogusław Jackowski, Janusz M. Nowacki, and Piotr Strzelczyk. 2001. "METATYPE1: A MetaPost-based engine for generating Type 1 fonts." EuroTEX 2001 Proceedings. Kerkrade, the Netherlands, 111–119.

[12] Włodzimierz Bzyl. 2001. "Re-introducing Type 3 fonts to the world of TEX." EuroTEX 2001 Proceedings. Kerkrade, the Netherlands, 219–243.

[13] Apostolos Syropoulos. 2000. "The MF2PT3 tool." Available online from `www.obelix.ee.duth.gr/~apostolo`.

[14] Rob Carter. 1997. *Experimental Typography*. A RotoVision Book. Watson Guptill Publications.

[15] František Muzika. 1965. *Die Schöne Schrift*. Verlag Werner Dausien, Hanau/Main. Vol I & II.

[16] Halina Thórzewska Ed. 2000. *More Precious Than Gold. Treasures of the Polish National Library*. Biblioteka Narodowa. Warszawa.

[17] Charlotte & Peter Fiell. 1999. *William Morris (1834–1896)*. Benedikt Taschen Verlag GmbH.

[18] J.R.R. Tolkien. 1981. *The Fellowship of the Ring*. Spółdzielnia Wydawnicza "Czytelnik". Warszawa.

[19] Donald E. Knuth. 1988. "A Punk Meta-Font". *TUGboat* **9**, 152–168.

[20] Richard M. Stallman and Roland McGrath. GNU *Make*. Available online as a part of GNU MAKE package.

[21] Per Cederqvist et al. *Version Management with* CVS. Available online with the CVS package. Signum Support AB.

[22] Mark Shoulson, 1994. *Okuda Font*. METAFONT source available online from `CTAN/fonts/okuda`.

[23] Karol Jarmakiewicz. 2002. *Czcionka Klingońska*. Instytut Matematyki, Uniwersytet Gdański.

[24] Mieszko Zieliński. 2002. *Kto i dlaczego wymyślił Tengwar*. Instytut Matematyki, Uniwersytet Gdański.

[26] Wojciech Górski. 2002. *Font Fenicki*. Instytut Matematyki, Uniwersytet Gdański.

[27] Sławomir Lis. 2002. *Pismo Ręczne*. Instytut Matematyki, Uniwersytet Gdański.

[28] Jacek Neuman. 2002. *Just Smiley!*. Instytut Matematyki, Uniwersytet Gdański.

[29] Alan M. Stanier. 1994. METAFONT source available online from `CTAN/fonts/iching`.

[30] Lesław Furmaga. 1999. *Ortofrajda. Pamieciowo-wzrokowy słownik ortograficzny dla dzieci*. INTEGRAF, Sopot.

[31] Jan Jelinek. 1977. *Wielki Atlas Prahistorii Człowieka*. Państwowe Wydawnictwo Rolnicze i Leśne. Warszawa.

FarsiTEX and the Iranian TEX Community

Behdad Esfahbod
Computing Center
Sharif University of Technology
Azadi Avenue
Tehran, Iran
farsitex@behdad.org
http://behdad.org/

Roozbeh Pournader
Computing Center
Sharif University of Technology
Azadi Avenue
Tehran, Iran
roozbeh@sharif.edu
http://sina.sharif.edu/~roozbeh/

Abstract

FarsiTEX, a localized version of LATEX, is a bilingual Persian/English typesetting package, meeting the minimum requirements of Persian mathematical and technical typography. This paper will describe FarsiTEX, together with its history, future and technicalities, its user community, and the reasons behind its success in Iran, amid its various usage and interoperability problems. It will also draw a general picture of the TEX community in Iran, and tries to describe why the community is still far from achieving its basic typographical needs.

Introduction

The Persian language, in its contemporary form, is a language spoken natively in Iran, Afghanistan, and Tajikistan. The local forms are known as Farsi, Dari, and Tajiki respectively. They all use the same basic vocabulary and grammar, but there are differences in both pronunciations and modern vocabulary. In this paper, we will focus on the form used in Iran, which is the official language of the country.

The modern Persian script, as written in Iran, is a right-to-left script with contextually changing shapes of letters, and a derivative of the Arabic script extended by addition of some letters (Peh, Tcheh, Jeh, and Gaf), and modification of a few others (Kaf and Yeh). The script, with roots in the Arab invasion of Persia in the 7^{th} century and later becoming known as the Perso-Arabic script, had then propagated to the areas currently known as Afghanistan, Pakistan, India, Western China, and then even South East Asia and Java, where many languages are written in it with further extensions to the alphabet, including Urdu, the official language of Pakistan. The Unicode Standard, in its latest version 3.2 (Unicode Editorial Committee, 2002), lists a total of 139 letters in the script, which are derivatives of about 28 basic Arabic letters.

The Persian typography, influenced by major calligraphic practices of the pre-printing era, is actually based on the famous Naskh style, which more than 99% of contemporary texts published in it. The alternate style, Nastaliq, a little harder to read but considered very beautiful by the general public, and widely known as the hardest commonly used script style in the world to implement in computers, has had a recent popularity after its many computer implementations appearing in the 1990s. But after a few years, because of readability problems, the usage of Nastaliq has been trimmed down to mainly school books on Persian literature.

Persian scientific typography, blossoming in the 1950s by publications of Gholamhossein Mosahab (who invented the *Iranic* font style, a back-slanted italic form to go with the right-to-left direction of the script), and Tehran University Press, that developed the means to publish the texts with the maximum achievable quality of the days. The human typesetters used many locally developed methods to extend the imported typesetting machines,

nowadays called "match stick methods", many of which used match stick parts to provide the proper spacings needed for mathematical formulas.

This was changed in late 1970s by the new typesetting machines made by Linotype which provided easier mechanisms for typesetting mathematics containing Persian text. The machines helped new publishers like Iran University Press and Fatemi Publishing Institute publish technical books in a much shorter typesetting period, making a large volume of mathematical books appear in the 1980s and early 1990s.

A leap happened in 1992, by appearance of two TEX-based typesetting packages called TEX-e-Parsi and LATEX-e-Farsi. The latter disappeared in a short while mainly because of incompetence, but the former, developed by Dadehkavi Iran with some investment from the two above-mentioned publishing houses, remained in use. TeX-e-Parsi design was highly influenced by the way Knuth had created TEX, doing thorough research on the existing typography of Iran at the time. The company, going bankrupt in 1997 because of high expenditure and limited market, has released the latest version of the package in 1996, based on pre-3.0 TEX and LATEX 2.09 with NFSS, but with various modifications both in the TEX engine and LATEX macros (SCO Unix and MS-DOS were supported as platforms). The package, still being used in a highly-tailored form by the mentioned publishing houses and a few mathematical departments, was unfortunately not affordable by individual authors and students. Thus, it could not help authors doing the document preparation themselves, and needed a special section in each department for typesetting manuscripts.

But the bigger leap was another package called Zarnegar, appearing in early 1990s for high quality typesetting using personal computers, which targeted the main stream of typesetting with various fonts and a visual markup language. Because of the good quality of the output and the reasonable price, the package got highly popular, and is still in wide use, estimated to be the second most popular document preparation software in Iran, after Microsoft Word. Unfortunately, Zarnegar's typesetting quality of mathematics is very poor, which has been a source of many badly-typeset technical books.

FarsiTEX

FarsiTEX started as an academic project by Mohammad Ghodsi in Computer Engineering Department of Sharif University of Technology. The project, known as FaTEX in the first year, started in 1991 as three BSc projects supervised by Ghodsi to provide the foundation (Haghghollahi, 1992; Asghari, 1993; Tajrobekar, 1993). After many experiments, FarsiTEX finally settled on the TEX--XET engine and the MS-DOS platform. The main work was done by Hassan Abolhassani and Mehran Sharghi in two MSc theses, the former working on a macro set with some ideas borrowed from the localized Hebrew version of LATEX 2.09 (Abolhassani, 1994), and latter on a METAFONT family of Persian fonts based on Linotron Badr, which he called Scientific Farsi (Sharghi, 1994). The contextual shaping of the letters was done by a pre-processor, which took input documents in the then widely used Iran System character set, and converted them to an internal code page which used four characters for each letter, each for one of the forms used in the Naskh style.

The system was in limited use by authors for about two years, until early 1996 when Ghodsi gathered a new team to concentrate on a public release of the software under GNU General Public License (Free Software Foundation, Inc., 1991). The team created a new syntax and character set for FarsiTEX input files, and consisted of Kiarash Bazargan, who created ftexed, an MS-DOS text editor based on Borland Turbo Vision, Mohammad Mahdian who wrote ftx2tex to handle the new file format, Roozbeh Pournader who revised the macro set, and Sharghi who revised his own fonts. The first public version appeared in October 1996, as an extension to emTEX distribution which was very popular at the time. Explicitly marked as beta-quality software, FarsiTEX was the first Iranian software released under GPL. A manual (Ghodsi and Pournader, 1997) was distributed with the package as a DVI file, and was also made available on paper for a very small fee.

FarsiTEX, imagined by its authors to have a very limited audience because of its scalability problems and various known bugs, grew rapidly among students and professors of mathematics, computer engineering, and physics all over the country, simply because it was the only affordable option available which was good enough for their basic typesetting tasks. The students, many of them now able to afford a PC at home, needed some software to run themselves. FarsiTEX was also evangelized by the new professors who had just returned to Iran after their studies in an American or European university and knew the value of document preparation by the author. Authors of FarsiTEX, betting on about a hundred users, were amused to find a base sized ten times that number.

The FarsiTEX Project Team, born in 1996 and still breathing amid various inactivity periods, has

released many small improvements since that time. Also, it has recently done a few alpha releases of a new system based on MikTEX, which includes a Microsoft Windows editor written almost from scratch (written by Mehrdad Sabetzadeh, Shiva Nejati, and Okhtay Ilghami), a localized version of *MakeIndex* supporting Persian ordering (by Nejati), and a FarsiTEX to HTML conversion program (by Mohammad Bakuii). It is unfortunate that the team has not released a single stable version yet, and the MS-DOS release is now frozen forever.

It may be worth noting that code contributions to the project from outside the project team has been very small, although there has been many serious users. The team members are still wondering about the possible reasons, but mostly blame it on the uncooperative nature of the Iranian people!

During these six years, the project has been financially supported by Sharif University of Technology, Ministry of Science, Research, and Technology, High Council of Informatics of Iran, Statistical Center of Iran, and Science and Arts Foundation.

TEXnical Details and Examples

Just like any other non-Unicode Persian software, FarsiTEX has its own character set, as unfortunately no 8-bit Persian character set has ever been both complete and popular. This character set and its inherent semantics make a special text editor an essential part of the FarsiTEX system, and the same time the major barrier for porting the system to other platforms, like Linux.

The Bidirectional Algorithm Bidirectionality, is the main issue to tackle in any Persian TEX system. The TEX--XET engine is of course capable of typesetting bidirectional text, but only if the directions are known *explicitly*. In other words, TEX--XET has nothing to do with the implicit directionalities of Unicode Bidirectional Algorithm (Davis, 2002) which, given some text in a logical order (a run of text as typed through a keyboard, for example) outputs the text in a visual order (the sequence of characters as should appear on a computer screen or a piece of paper). This mapping is far from trivial in cases that characters of both directionalities mix with *neutral* characters (like punctuations and spaces), or weakly directioned ones (like digits).

In FarsiTEX, the text editor is responsible for converting the logical order to the visual one. The editor manipulates files with the ftx extension, which are in a special semi-logical semi-visual bidirectional format designed to be as near as possible to the internal representation of the editor (which is in visual order). This format has simplified the bidirectional algorithm by using two different codes for many neutral characters like space and parentheses, one for each of the left-to-right and right-to-left modes. The idea of having different characters for different directions has been borrowed from the ISIRI 3342 (Institute of Standards and Industrial Research of Iran, 1993), a national Iranian character set standard.

The ftx format, while easy to process for the editor, is not suitable for a TEX-like engine, which raises the need for the ftx2tex converter, that reorders the visual text in the ftx file to the logical order, explicitly marking the directionality using `\InE` (Insert English), `\EnE` (End English), `\InF` (Insert Farsi), and `\EnF` (End Farsi) macros. These macros enable the engine to typeset a text in both directions.

A screenshot of the Microsoft Windows editor, is shown in Figure 1 (FarsiTEX's output can be seen in Figure 2). Two different background colors are used to specify the characters' direction, needed for neutral characters like space, full stop, and parentheses. So, unlike the common bidirectional algorithms, and thanks to the background color, there are no ambiguities in the direction of neutral characters. But the problem of nesting different directionalities still remains.

Joining, Shaping, and Line Justification The Persian script, being a derivative of Arabic, is a cursive script, which means that two adjacent letters may *join* to each other, forming up to four different glyphs for each letter. The ftx2tex converter is responsible for detecting the pairs that join (*the joining algorithm*) and selecting the proper glyphs based on joining information (*the shaping algorithm*).

When a typesetter is justifying the lines in a Persian paragraph, it is common to stretch the joining line that appears between two adjacent glyphs. There is no inter-letter spacing in FarsiTEX, and only the joining stem will be stretched. To implement this behavior, the ftx2tex converter inserts a *stretchable kashide* character (also known as *tatweel*) character between the two connected letters. This inserted character is defined as an active character expanding to a horizontal glue filled by horizontal rules. A sample of the behavior can be seen in Figure 3.

FarsiTEX Forever

The FarsiTEX Project Team is currently working on a new release with PostScript Type 1 fonts, moved by the serious need of the user community

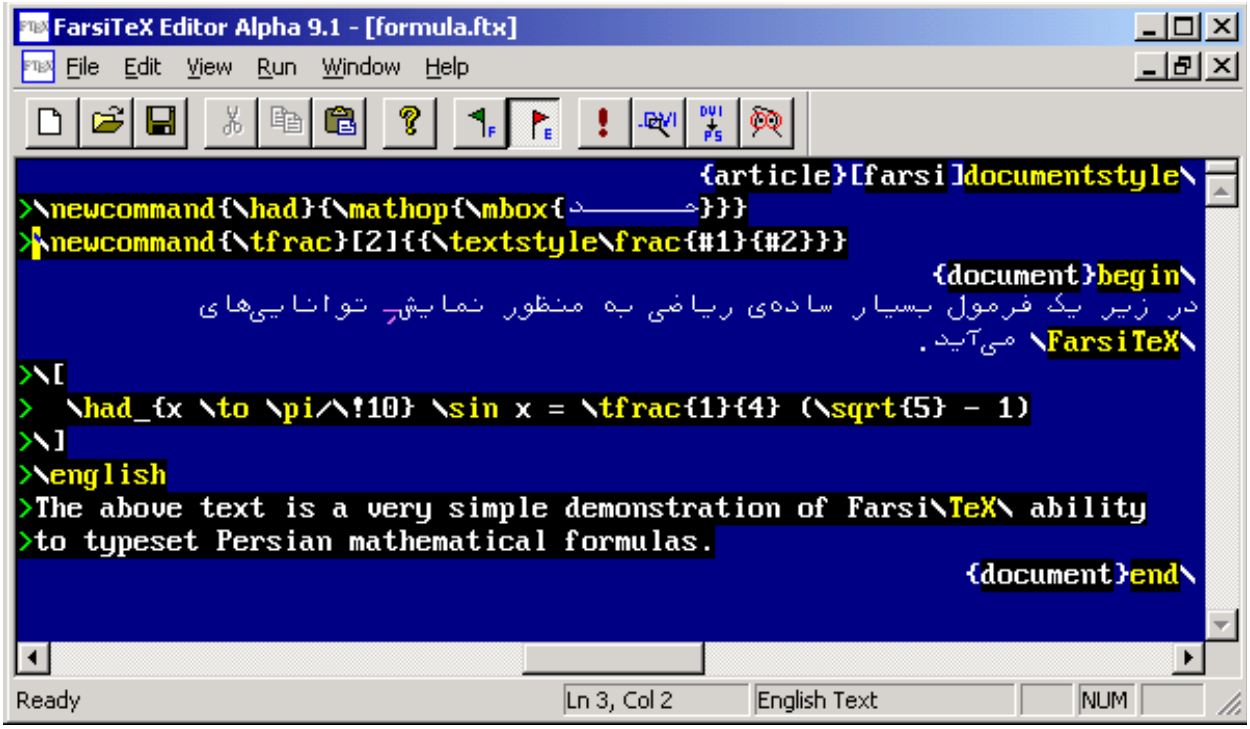


Figure 1: The FarsiTEX editor running under Microsoft Windows. Notice the background color of backslashes and curly braces in the right-to-left lines.

در زیر یک فرمول بسیار ساده‌ی ریاضی به منظور نمایشِ توانایی‌های فارسیTEX می آید.

$$\lim_{x \to \pi/۱۰} \sin x = \tfrac{۱}{۴}(\sqrt{۵} - ۱)$$

The above text is a very simple demonstration of FarsiTEX ability to typeset Persian mathematical formulas.

Figure 2: FarsiTEX's output, with the input given in Figure 1. Notice the automatic replacement of European digits (also known as Arabic digits) by Persian ones. The operator appearing before the sine is an alternate form of `\lim`, used in high school mathematics textbooks in Iran.

اَلا یـا اَیُـهَـا الـسّـاقـی اَدِر کَـأسـاً وَ نـاوِلْـهـا — که عشق آسان نمود اول ولی افتاد مشکل‌ها

به بوی نافه‌ای کاخر صبا زان طره بگشاید — ز تابِ جعدِ مشکینش چه خون افتاد در دل‌ها

مرا در منزلِ جانان چه امنِ عیش چون هر دم — جرس فریاد می‌دارد که بر بندید محمل‌ها

به مِی سجاده رنگین کن گرت پیرِ مغان گوید — که سالک بی‌خبر نبود ز راه و رسمِ منزل‌ها

شبِ تاریک و بیمِ موج و گردابی چنین هایل — کجا دانند حالِ ما سبک‌بارانِ ساحل‌ها

همه کارم ز خودکامی به بدنامی کشید آخر — نهان کِی ماند آن رازی کز او سازند محفل‌ها

حضوری گر همی‌خواهی از او غایب مشو حافظ — مَتیٰ ما تَلْقَ مَنْ تَهویٰ دَعِ الدُنْیا وَ اَهْمِلْها

Figure 3: A sonnet by Hafez, typeset in two columns with text stretched for equal width. This style is necessary for typesetting traditional poems, where justification in shape was a visual reference to the poem's rhyme.

to publish their documents in PDF, and also a Linux text editor, which will make the first teTEX-based Linux release possible. Other plans include Unicode support and integration with Omega, which will need a complete review of the system. The project is being continued in Computing Center, Sharif University of Technology, and can be reached at

`http://www.farsitex.org/`

(which is hosted at SourceForge.net).

References

Abolhassani, Hassan. *Typesetting Persian Documents using TEX*. Master's thesis, Computer Engineering Department, Sharif University of Technology, 1994.

Asghari, Parvaneh. "Scientific design of Traffic fonts using METAFONT". BSc project report, Computer Engineering Department, Sharif University of Technology, 1993.

Davis, Mark. "The Bidirectional Algorithm". Unicode Standard Annex #9, The Unicode Consortium, 2002. Available from `http://www.unicode.org/unicode/reports/tr9/`.

Free Software Foundation, Inc. "GNU General Public License (GPL)". 1991. Available from `http://www.gnu.org/licenses/gpl.html`.

Ghodsi, Mohammad and R. Pournader. *The FarsiTEX Manual*. Computer Engineering Department, Sharif University of Technology, 1997.

Haghghollahi, Jafar. "Designing Persian fonts using METAFONT". BSc project report, Computer Engineering Department, Sharif University of Technology, 1992.

Institute of Standards and Industrial Research of Iran. "ISIRI 3342, Farsi 8-bit Coded Character Set for Information Interchange". 1993.

Sharghi, Mehran. *Scientific Design of Persian Fonts*. Master's thesis, Computer Engineering Department, Sharif University of Technology, 1994.

Tajrobekar, Laleh. "Designing with PostScript in Persian environments". BSc project report, Computer Engineering Department, Sharif University of Technology, 1993.

Unicode Editorial Committee. "Unicode 3.2". Unicode Standard Annex #28, The Unicode Consortium, 2002. Available from `http://www.unicode.org/unicode/reports/tr28/`.

The Marriage of TEX and Lojban

Hong Feng
Suite 3-3, 200 WuZhong Str.
Wuhan, Hubei Province
430040 China P.R.
hongfeng@gnu.org

Abstract

Lojban is an old artificial language which is ambiguity free, it can also be used as a tool to express Chinese text encoding in readable ASCII characters, and thus be applied in TEX typesetting system. Keywords: TEX, Lojban, Chinese.

When Prof. Donald Knuth invented the TEX system, Chinese was not considered as the default language to support, as TEX only accepts the readable 7-bit ASCII characters. Chinese, either the simplified, or the traditional encoding set, has many thousands of characters. For example, the GB2312-80 contains 6,763 simplified Chinese characters, which requires at least double-byte (16-bit) to represent one Chinese character (the encodings in the double-byte format are unreadable for people unless one checks the encoding table one by one!), and makes it difficult for TEX to typeset Chinese documents.

Various scenarios were presented to support Chinese and many relevant macros were developed in the past, such as those in LATEX and ConTEXt and the CJK package developed by Werner Lemberg, which is distributed with the TEX Live CD. The new Omega system tries to work directly with Unicode, which is a popular standard using 16-bit encoding. Despite the differences in the technical implementation details, all of them have assumed that Chinese characters are implicitly expressed in the fixed-length double-byte encoding which are not readable by people.

CTUG (Chinese TEX User Group) is trying another completely different approach to work out this problem. By discarding the man-made implicit assumption of the fixed character length in double-byte, CTUG imported Lojban to represent Chinese encoding in variable length but still in the readable ASCII set. This paper documents the idea in some detail, and points out the future tasks to do under the scenario.

Background Information about Lojban

Lojban (pronounced as LOZH-bahn), which stands for "Logic Language" in Lojban, is nothing new; actually it was presented as a constructed language in 1955 with the name "Loglan" by the project founder Dr. James Cooke Brown. It is based on the "Sapir-Whorf" hypothesis, which states that the structure of a language constrains thought in that language, and constrains and influences the culture that uses it. Over more than the past four decades, Lojban has become a mature artificial language. Here we highlight the main features of Lojban:

- Lojban is designed to be used by people in communication with each other, and now also possibly with computers.
- Lojban is designed to be culturally neutral. It is based on fully phonetic spelling, so people can learn to read Lojban on the fly.
- The regular grammar of Lojban is based on the principles of logic, which has an unambiguous grammar, and has successfully passed the YACC testing. This removes restrictions on creative and clear thought and communication.
- Lojban is designed as a simple language, with just 1,300 root words. Using these root words, it is possible to combine and form millions of words in a vocabulary with ease.

In essence, Lojban is quite close to Chinese grammar, thus a Chinese can quickly become a Lojban user. In the training seminar given by CTUG, practice has shown that some CTUG members could learn and grasp it within a week.

Chinese as Expressed in Lojban

Now, let's check how Chinese words are constructed. Overall, most Chinese linguists have agreed that Chinese has only four methods to construct a character: *XiangXing, ZhiShi, HuiYi* and *XingSheng*. Although it is hard to describe them in formal language, any Chinese character is constructed by one of these four methods, and the first method *"XiangXing"* is fundamental to the construction.

XiangXing means drawing a sign for a given meaning; thus Chinese is classified as an ideograph system in the language taxonomy.

According to researches into the signs recorded in ancient tortoise bones, the most original and frequently used signs are fewer than 500. Tortoise bones are the back shell of tortoises. Chinese people recorded the oracle on them. The signs of the oracle are the oldest Chinese characters we have discovered so far, and they are the origin of modern Chinese characters. And statistically, Chinese characters constructed by *XingSheng* (mostly based on *XiangXing* ideographs) occupy more than 90% of the modern Chinese character repertoire.

A character of *XingSheng* consists of two parts: one part indicates the pronouncation of the character, and the other part indicates the meaning of the character.

For example, my name in Chinese, Hong Feng, is expressed in two characters; each of them is constructed as a *XingSheng* character. Hong has two parts: the three points at the left side is the *Xing* part, and indicates the character is related to water; the right part is the *Sheng* part, pronounced as "gong", meaning the character has "ong" in the pronounciation. The character means large-scale, macro, magnificent, giant.

Feng has two parts too. The left part is also the *Xing* part, a *XiangXing* ideograph for mountains, and the right part specifies the pronouncation to be "feng". The character means the top of the mountain.

Thanks to the more than five thousand years of the simplification movement in the history, the grammar rules of the language are truly simple today. Chinese texts are very similar to an assembly line which we have seen in an automobile production workshop — Chinese characters are placed one by one without stop (i.e. without blank space left between them), quite like the TEX places characters in a box one after another to form a word, and words are placed one after another to form a sentence or a line, and lines are placed one after another to form a paragraph, and paragraphs are placed to be a page to the end.

In the TEX system, if you have a new sign which is not defined yet, then you could design the glyph of the new sign in METAFONT (or in the PostScript language) in a box, and give the box a name (as a new control sequence) to the METAFONT (or to the PostScript) program; after doing that, you could use the new sign with TEX, as if it were one of the built-in readable ASCII characters.

Such cases have happened many times in TEX history. For example, the Euro currency was put into use on January 01, 2002, but the currency symbol was made available much earlier for TEX by two NTG members, who designed the symbol in METAFONT (see MAPS Number 27), so now you could express the Euro in a TEX document directly by `\symbol[euro]`.

Likewise, Chinese characters can be handled in the same way, and once we give each sign in a box a name in Lojban (which also means we discarded the fixed double-byte Chinese encoding, instead, we use variable length of the readable ASCII characters to represent the Chinese, then TEX can be regarded as a native formatter for Chinese immediately!

If we design carefully, we can build a one-to-one mapping table between the existing Chinese encoding set (GB, Big5, Unicode or whatever) and Lojban the expression set, which makes the conversion easy to handle by scripts in Perl or whatever. As Lojban expressions are in readable ASCII, they can be edited using any editor (such as GNU Emacs) even on a simple text-only terminal.

Marriage of TEX and Lojban

As we have seen above, it is possible to encode Chinese by using Lojban as the meta-language. This is the key step to get marriage of TEX and Lojban to happen.

It is necessary to review how TEX defines a control sequence. In TEX, π is defined as `\pi`, likewise, supposing we defined the glyphs of Chinese figures (from zero up to nine) in control sequences in Lojban respectively like this:

```
Chinese   Lojban
=================
zero      \no
one       \pa
two       \re
three     \ci
four      \vo
five      \mu
six       \xa
seven     \ze
eight     \bi
nine      \so
```

Now, we can typeset the Chinese number "two zero zero two" this way in TEX: `\re\no\no\re`; the backslash characters won't add too much burden for

people to read, and by designing a new macro, it is possible to remove them like this:

`\chinese{re no no re}`.

`TUG‘‘two zero zero two’’` (English and Chinese combined together for TUG2002) can be represented in `TUG\chinese{re no no re}`.

TEX and Lojban have agreed to marry.

Tradeoffs and Benefits

The tradeoff of the marriage is obviousl: it adds one step to convert the current Chinese double-byte encodings into Lojban expressions, though we can let a computer do the job automatically, which can be counted as a trivial matter.

Perhaps the true tradeoff would be the requirement for the TEX user to understand more or less about Lojban, thus enlarging the learning curve. Though it is not indispensable, the more one knows Lojban, the higher efficiency would be obtained during the typesetting. If one can read and write in Lojban, then it is possible to typeset Chinese directly in TEX (without the conversion mentioned as above).

Now let's check the benefits. We can just use the word "tremendous" to describe so many benefits we will have under this scenario. Chinese expressed in Lojban is in human readable ASCII characters, which are supported in almost any computer nowadays. This means, all existing programming languages can be used to support Chinese directly too. There are also other benefits to describe the XML's meta data for Chinese in Lojban (again, it is unambiguous in grammar), which goes beyond the scope of this paper.

To-do tasks

To make this scenario become reality, we have several major tasks to do:

- Define a Chinese-Lojban dictionary. As mentioned above in section 2, there are approx. 500 ideographs to define, and it also requires definition of three other methods in Lojban. Lojban is suitable to describe the logic relations because it is designed as a logic language.
- Lojban specification comes with a dictionary which contains ca. 1,300 root words, so it just requires some time and care to build the mapping relation to finish the task.
- Define free, high quality fonts of Chinese. In practice, at least four fonts are required. Now this is a part of the MNM Project (MNM's Not Millions). There are many non-free Chinese fonts available, so commercial publishers can purchase the non-free fonts, and we can add the fonts by applying this scenario.
- Mapping the Chinese fonts adds value to the control sequences in Lojban as key in a hash table. Once the hash table is ready, we could build it into the TEX source tree. The system will be ready to use.

Conclusion

This paper only explained a very small part of the power of Lojban — how to typeset Chinese in TEX using another approach. By importing the idea of re-encoding Chinese in variable length strings of human readable ASCII codes, instead of fixed length in double-byte, the TEX system without any modification can support Chinese typesetting directly.

About the Author: Hong Feng is the founder and the Chairman of the Chinese TEX User Group. He is also the co-founder of the FSF-CHINA Academy. He can be reached by `hongfeng@gnu.org`

References

[1] Donald E. Knuth. *The TEXbook*. Addison-Wesley, Reading,Massachusetts, 1984.

[2] Hans Hagen. *The euro symbol*. MAPS, Number 27, 2002.

[3] `http://www.lojban.org`. *Lojban Reference Grammar*.

INVITED KEYNOTE TALK

ConTEXt, XML and TEX: State of the Art?

Hans Hagen
Pragma, Netherlands
pragma@wxs.nl

Abstract

The ConTEXt macro package was originally written to ease the development of complex and educational documents. Over time, more and more features were added and a tight integration with METAPOST has been achieved. A few years ago, ConTEXt became XML-aware, so that users can now comfortably mix XML and TEX techniques.

In spite of what the XML community preaches and DTP applications promise, typesetting complicated documents is still far from trivial. To some extent this has become easier, but the basic problems stay the same. ConTEXt tries to bridge those worlds by supporting modern (XML driven) workflows, by providing a programmable (TEX based) typesetting environment, and by offering high quality (DTP competitive) output.

Currently, its authors use ConTEXt in a wide range of applications:

- automatic typesetting of highly structured input into relatively complex layouts, either under the authors control or not;
- implementing publishing on demand workflows where ConTEXt acts in the background;
- dynamic generation of documents based on user input or data bases (internet applications);
- special applications where PDF is used as user interface (workflow optimization).

Although TEX is over twenty years old, it is still a strong player in the field of document processing. In this presentation we hope to illustrate that TEX can meet many of the demands of today's publishing.

Low-level Devanāgarī Support for Omega — Adapting devnag

Yannis Haralambous
Département Informatique
École Nationale Supérieure des Télécommunications de Bretagne
BP. 832, 29285 Brest, France
yannis.haralambous@enst-bretagne.fr
http://omega.enstb.org/yannis

John Plaice
School of Computer Science and Engineering
The University of New South Wales
UNSW Sydney NSW 2052, Australia
plaice@cse.unsw.edu.au
http://www.cse.unsw.edu.au/school/people/info/plaice.html

Abstract

This paper presents tools (OTPs and macros) for typesetting languages using the Devanāgarī script (Hindi, Sanskrit, Marathi). These tools are based on the Omega typesetting system and are using fonts from devnag, a package developed by Frans Velthuis in 1991. We are describing these new OTPs in detail, to provide the reader with insight into Omega techniques and allow him/her to further adapt these tools to his/her own environment (input method, font), and even to other Indic languages.

सारांश

यह लेख देवनागरी लिपि को प्रयोग करते हुए भाषाओं की टाईपसैटिंग या लेखायोजन के लिए प्रयोग किए जाने वाले टूल को प्रस्तुत करता है। ये टूल ओमेगा टाईपसैटिंग पर आधारित है और देवनाग के लेखारुपो को प्रयोग करता है जोकि फ्रांस वेलथुइस द्वारा 1991 मे बनाया गया पैकेज है। हम इन ओ टी पी को विस्तृत कर रहे हैं तांकि पक्ऋहने वालों को ओमेगा तकनीक की जानकारी हो जाए और वो अपने आप को इस टूल से अपने वातावरण और दूसरी भारतीय भाषाओ के अनुरुप बदल सके या ढ़ाल सके।

Introduction

One of the first Indic language support packages for TEX was devnag, developed by Frans Velthuis in 1991.[1] At that time it was necessary to use a preprocessor for converting Hindi or Sanskrit text written in a way legible to humans into data legible by TEX. This preprocessor allowed the use of an ASCII transcription, and performed the contextual analysis inherent to Devanāgarī script, as well as pre-hyphenation (by explicitly inserting hyphenation points). The preprocessor was necessary for two main reasons:

1. A Sanskrit font contains over 300 glyphs, when ligatures are taken into account.
2. The TFM and VF languages are not powerful enough to make all the necessary glyphs out of a font of 256 characters.

Using a preprocessor has many disadvantages, due mainly to the fact that it has to read not plain text, but rather LATEX code. It also has to avoid treating commands and environment names as Devanāgarī text. So the preprocessor should be clever enough to distinguish text from commands, i.e., content from markup.

It is well known that, in the case of TEX, this is practically impossible, unless the preprocessor is TEX itself (there is a notorious saying: "only TEX can read TEX").

So much for computing in the 20^{th} century. Nowadays we have other means of processing information, and the concept of (external) preprocessor

[1] A second system for processing Devanāgarī was created by Charles Wikner. It has important features lacking in Velthuis's devnag system, but unlike the latter it did not address the setting of Hindi text. The general design of the system – Metafont plus pre-processor – was identical to that of Velthuis.

is obsolete. In fact, the same operations are done inside Omega, a successor of TEX. Processing text internally has the crucial advantage of allowing the processor to distinguish precisely what is content and what markup (at least as precisely as TEX itself does it).

This makes it much easier to treat properties inherent to writing systems: one only needs to concentrate on the linguistic and typographical properties of the script, and one doesn't need to think of what to "do with LATEX commands" in the data stream.

Furthermore, there is an efficiency issue: using Omega there is only one source file, namely the TEX file (and not a pre-TEX file and a TEX file); one doesn't need to care about preprocessor directives; the system will not fail because of a new LATEX environment which is not known to `devnag`; mathematics and other similar constructions do not interfere with Devanāgarī preprocessing.

Contextual analysis of Devanāgarī script has, at last, become a fundamental property of the system, independent of macros and packages.

Unicode and Devanāgarī

The 20-bit information interchange encoding Unicode (`www.unicode.org`) has tables for all Indic writing systems, based on a common scheme (so that phonetically equivalent letters are placed on the same relative positions in each table). The first of these tables (positions `0900-097F`, see Table 1) covers Devanāgarī.

For historical reasons (compatibility with legacy encodings) the Unicode approach to Devanāgarī is quite awkward: it is partly logical and partly graphical. For example, there are separate positions for independent and dependent versions of vowels: when encoding text one has to choose if a given vowel is dependent or independent, although this clearly derives from contextual analysis, as in Velthuis' transcription where both versions of vowels have the same excellent input transcription.

On the other hand, this method is not applied to consonant *ra*; indeed, placing a *ra* before a cluster of consonants is graphically represented by a mark on the last of the consonants (compare क्त *kta* and र्क *rkta*) — this mark is not provided in the Unicode table, and hence application of this feature is left to the rendering engine.

Nevertheless, despite its weaknesses, Unicode is very important because it ensures compatibility between devices all around the world: a text written in Devanāgarī and encoded in Unicode can be processed (read, printed, analyzed) on every machine or software that is Unicode compliant.

Omega fullfils Unicode compliance, and the system we are describing in this paper is designed in such a way that Unicode-encoded texts can be processed equally well as texts encoded in Velthuis' transcription.

Installation and Usage

The Omega low-level support[2] of Devanāgarī consists of eight OTPs (Omega Translation Processes) and a small file with macros:

```
velthuis2unicode.otp
hindi-uni2cuni.otp
hindi-uni2cuni2.otp
hindi-cuni2font.otp
hindi-cuni2font2.otp
hindi-cuni2font3.otp
sanskrit-uni2cuni.otp
sanskrit-cuni2font.otp
odev.sty
```

OTP files have to be converted to binary form (`*.ocp`) and placed in a directory where Omega expects to find them.

To typeset text in Devanāgarī, use the commands `\hindi` or `\sanskrit` (depending on the language of your choice) inside a group, and keyboard the text in Velthuis' transcription (see Table 1, taken from Velthuis' `devnag` documentation[3]). For example, `{\hindi kulluu, acaanak, \sanskrit kulluu, acaanak}` will produce कुल्लू, अचानक, कुल्लू, अचानक्.

Description of the OTPs

This description is a bit technical and demands both some knowledge of Omega and of Devanāgarī script. The reader can find more information on the former, on the Omega Web site[4] and on the latter in books about Devanāgarī script. In particular, there is a very nice introduction to the contextual features of the script in the Unicode book[5] (Section 9.1).

[2] We call it "low-level," because there is no standard LATEX3-compliant high-level language support interface yet. We don't know yet how languages and their properties will be managed in LATEX3 and therefore do not attempt to introduce yet another syntax for switching to Hindi or Sanskrit or Marathi. Instead, we — temporarily — use a `devnag`-like syntax: simple commands `\hindi` and `\sanskrit` which have to be placed inside groups, as in the good old days of `plain` TEX...

[3] To be found on CTAN, `language/devanagari/distrib/manual.tex`.

[4] `http://omega.enstb.org`

[5] The Unicode Standard, Version 3.0, Addison Wesley, Reading Massachusetts, 2000.

velthuis2unicode.otp In this OTP we convert Velthuis' input transcription into Unicode. It is a quite short OTP (about 80 lines), with lines of the type

```
`z' => @"095B @"094D ;
`a'`a' => @"0906 ;
```

On the second line, the pair of letters `aa` of Velthuis' transcription is sent to Unicode character `@"0906` (independent vowel "aa"). On the first line, letter `z` is sent to Unicode characters `@"095B` (letter "za") and `@"094D` (virama).

This may seem strange, but indeed the plan is to convert in a later step independent vowels into dependent ones, and to use the virama as a way to find out if a given consonant is part of a consonantic cluster or not. This will be done in the forthcoming OTPs.

(sanskrit|hindi)-uni2cuni.otp In this OTP we deal with virama and dependent vowels. First of all, in the case of Hindi, we remove the (possible) final virama of the word:

```
{CONSONANT} {VIRAMA} end: => \1 ;
{CONSONANT} {VIRAMA} {NONHINDI} =>
  \1 <= \3 ;
```

In these two lines, we remove virama which may be either at the end of the input buffer, or before a non-Hindi character. In the latter case, the non-Hindi character is put back in the stream. The code above is for Hindi. In the case of Sanskrit, we add a (fake) Unicode character which will represent internally the final virama:

```
{CONSONANT} {VIRAMA} end: => \1 @"097F ;
{CONSONANT} {VIRAMA} {NONHINDI} =>
  \1 @"097F <= \3 ;
```

Follow lines of the type:

```
{CONSONANT} {VIRAMA} {INITA} => \1 @"097D ;
{CONSONANT} {VIRAMA} {INITAA} =>
  \1 @"093E @"097D ;
```

Indeed, by placing a virama systematically after each consonant, we have also added viramas between consonants and vowels, which makes no sense. On the first line, the 'short a' vowel is removed together with the (spurious) virama. On the second line, the 'long a' vowel is replaced by Unicode character `"@"093E`, which is the dependent version of vowel 'long a', and the virama is removed. There are such lines for each vowel.

Notice the presence of "fake" Unicode character `@"097D`. This character will be replaced by a soft hyphen at the very last step of our OTP chain.

A special case is the vowel 'short i', where the glyph representing it has to be placed in front of the consonantic cluster. This is done by lines of the type:

```
{CONSONANT} {VIRAMA} {INITI} =>
  @"093F \1 \2 @"097D ;
```

where we have a consonant and virama followed by a 'short i' vowel. In this case we place Unicode character `@"093F` followed by the consonant. On similar lines, we have n-uplets ($n \leq 7$) of consonants and viramas followed by a 'short i' vowel; we replace them by `@"093F` followed by the group of consonants and viramas, except for the last virama.

hindi-uni2cuni2.otp One thing that has not been covered by the previous OTPs is the case of consonantic clusters starting with an 'r' consonant: in this case, a mark is placed on the last consonant ofthe cluster. This mark is not part of the Unicode encoding, and hence we have to use a fake Unicode character. This file contains lines of the type:

```
{RA} {VIRAMA} {CONSONANT} => \3 @"097E @"097D ;
```

On this line we replace a consonant preceded by a 'ra' and a virama, by the same consonant but followed by the fake Unicode character `@"097E` which will be replaced in the next OTP by the TEX code producing the mark we need.

(sanskrit|hindi)-cuni2font.otp In this OTP, which is quite long (328 lines), we start switching from Unicode to font encoding: this specific file — as well as files `cuni2font2.otp` and `cuni2font3.otp` — deals with `devnag` font encoding, but the user can write his/her own files for a different font encoding[6].

First of all we define aliases for all consonants, to make the writing of ligature expressions easier:

```
aliases:

BA = (@"092C) ;
BHA = (@"092D) ;
...
VIRAMA = (@"094D) ;
```

Then we write the ligature expressions, using expressions like the following:

```
{SSA} {VIRAMA} {TTA} {VIRAMA} {YA} => @"00F7 ;
{SSA} {VIRAMA} {TTA} {VIRAMA} {VA} => @"00AB ;
{SSA} {VIRAMA} {TTA} {VIRAMA} {RA}
  {VIRAMA} {YA} => @"00AA ;
{SSA} {VIRAMA} {TTA} {VIRAMA} {RA} => @"0104 ;
```

As the reader can see, the virama is used to ensure that these consonants are indeed part of the

[6] For example for the prestigious *Monotype Devanagari* (`http://www.agfamonotype.com`) which is, IOHO, one of the most beautiful existing fonts, and has even pre-designed glyphs for consonants with dependent short and long 'u' vowels.

same consonantic cluster. Since in OTP files order of precedence is based on order of expressions in the expression list, the fact that line 3 is before line 4 ensures that the consonantic cluster .s.try is indeed detected instead of .s.tr, which will be matched only if the last letter is not a y.

Notice that our right-hand expressions are already in the devnag font positions, except for the one of the last line (@"0104), which is a 'fake' glyph position — like we had previously 'fake' Unicode characters — i.e., a byte which will be detected by a forthcoming OTP and converted into something that makes sense.

The devnag system allows preprocessor directives activating and de-activating individual ligatures; we do not have an equivalent feature in our system because we do not consider it to be crucial. Instead, the user has the possibility to add or remove lines as the ones above in the OTP file, save the OTP under a different name and use it as a replacement of the standard one, or in a new OCP list. In the latter case, one can switch on-the-fly from one ligature setup to another.

After the ligature expressions, follow the consonants followed by virama:

```
{BA} {VIRAMA} => @"004E ;
{BHA} {VIRAMA} => @"003C ;
{CA} {VIRAMA} => @"0051 ;
{CHA} {VIRAMA} => "\qq{" @"0043 "}" ;
{DA} {VIRAMA} => "\qq{" @"0064 "}" ;
...
```

If one of these patterns is matched, this means that (a) we are inside a consonantic cluster and (b) all ligatures have been matched. Two options remain: either there is a special half-form of the glyph of the consonant, or an explicit virama is placed under the normal version of the consonant glyph.

Using "halfed" glyphs is, in a sense, intermediate between predefined ligature glyphs and the placement of individual "independent" glyphs next to each other. It is a method to construct arbitrary ligatures using the basic letter part: compare, for example, for the same consonantic cluster "vva," व्व (Sanskrit ligature), व्व (Hindi ligature, where the first व is half-form) and व्‌व (two individual consonants, the first having a virama).

In the code above, macro \qq inserts the virama. In version 2 of Omega this macro will be made obsolete, since placement of diacritics will be handled by μ-engines; until then, we use macros, like \qq, taken from the devnag package.

Follow two special lines:

```
{RA} {DEPU} => @"007A ;
{RA} {DEPUU} => @"0021 ;
```

where DEPU and DEPUU stand for "dependent short u" and "dependent long u." These cover the special glyphs for consonant 'ra' with these vowels: र + उ → रु, र + ऊ → रू.

Finally, follow lines of the type:

```
{BA} => @"0062 ;
{BHA} => @"0042 ;
{CA} => @"0063 ;
...
```

which simply match consonants (with inherent "short a" vowel) and glyph positions, as well as lines like

```
{CANDRABINDU} =>
  "\llap{{\clearocplists\char32}}" ;
{ANUSVARA} =>
  "\llap{{\clearocplists\char92}}" ;
{DEPAI} =>
  "\llap{{\clearocplists\char123}}" ;
...
```

which map characters candrabindu, anusvāra, dependent "ai" vowel and similar signs with the necessary TEX code to obtain their glyphs. Once again this code will be obsolete in Omega v.2.

hindi-cuni2font2.otp This OTP, as well as the next one, are provided to deal with cases which could not be handled simultaneously with the previous one. The present file deals with dependent vowels "short u," "long u," "short r," "short l," "English o," which have the common property of being centered under the letter. Until Omega 2 arrives, we need to use macros to place them, and these macros have to be placed *before* the consonant which carries the vowel, so that this consonant can be their argument:

```
(@"0000-@"00FF) {DEPU} => "\qqqa{" \1 "}" ;
(@"0000-@"00FF) {DEPUU} => "\qqqb{" \1 "}" ;
(@"0000-@"00FF) {DEPR} => "\qqqc{" \1 "}" ;
(@"0000-@"00FF) {DEPRR} => "\qx{" \1 "}" ;
(@"0000-@"00FF) {DEPL} => "\qy{" \1 "}" ;
(@"0000-@"00FF) {DEPLL} => "\qz{" \1 "}" ;
(@"0000-@"00FF) {DEPOO} => "\qzz{" \1 "}" ;
```

We could not obtain them in the previous OTP, since that file matched the Unicode characters of consonants and replaced them with font positions. Of course we could include in that file *combinations* of consonants and vowels, but this would make the file unnecessary long: it is easier to match first the consonants and, at a second stage, the vowels.

Since we are now matching font positions, the left-hand expressions use `(@"0000-@"00FF)`. This works only because dependent vowels always follow consonants. Nevertheless it is not very elegant, and this code will return to non-existence as soon as μ-engines are available.

hindi-cuni2font3.otp This very short file (25 lines), deals with the final virama and with some special cases of clusters: combinations of various consonants and consonant "ra." The final virama is used in Sanskrit only (according to Velthuis' convention) and has to be dealt with separately because in our OTP system we have used the (regular) virama as a marker of consonants *inside* consonantic clusters. If we had used a "regular" virama also at the end of the word, then we would obtain consonantic clusters with only half-forms of consonants and no full-form at the end. Instead, we have used a fake Unicode character (`@"097F`) for the final virama and are replacing it by its TEX code only at the very last step, as follows:

```
(@"0000-@"00FF) @"097F => "\qq{" \1 "}";
```

It can happen that the last consonant, although it has no vowel, carries a special sign because the consonantic cluster starts with consonant "ra;" to handle that case we have two extra lines:

```
(@"0000-@"00FF) @"097E @"097F => "\qq{"
  \1 "}\llap{\clearocplists\char13}" ;
(@"0000-@"00FF) @"097F @"097E => "\qq{"
  \1 "}\llap{\clearocplists\char13}" ;
```

which send the two combinations of fake Unicode characters to the same TEX code producing both the "ra" mark and the virama.

Finally there is a line replacing the fake Unicode character `@"097D` used to temporarily stand for the soft hyphen, with the adequate `\discretionary`:

```
@"097D => "\discretionary{\hyph}{}{}" ;
```

If the user does not wish hyphenation, he/she can replace this line by a simpler one, which will "absorb" all `@"097D` characters:

```
@"097D => "" ;
```

Conclusion

The purpose of the previous sections was to illustrate the processing of a given script (Devanāgarī) by Omega Translation Processes. Omega 2 will make these even more efficient since code used to center diacritics under consonants will be replaced by μ-engines. We believe that these methods can be applied to other Indic scripts. Furthermore the fact of having three kinds of files:

1. OTPs for converting input transcription into Unicode;
2. OTPs for handling contextual analysis of Indic scripts;
3. OTPs for converting real or fake Unicode characters into glyphs for a given font,

make this system easily adaptable to any combination of input method and font.

We invite Omega users around the world to write the necessary code and make it available to other through CTAN servers. We hope that these tools will make processing of Indic languages easier and more efficient and will allow production of high quality documents.

Availability and Thanks

All resources described in this document are free software and are available on CTAN. The OTP and macro files described in this paper can be found on CTAN in `language/devanagari/omega`

We invite users writing software for Omega typesetting of Indic languages into similar `omega` directories inside the corresponding Indic language directories.

The authors would like to thank Anish Mehta and Gagan Sharma, *stagiaires* at ENST Bretagne at the time this paper was written, for their valuable help.

Figure 1 on page 55 reproduced by kind permission of the Unicode Consortium.

0900 **Devanagari** **097F**

Figure 1: Unicode Table for Devanāgarī Script

Reproduced by permission of the Unicode Consortium

Vowels											
a	अ	—	aa, A	आ	ा	i	इ	ि	ii, I	ई	ी
u	उ	ु	uu, U	ऊ	ू	.r	ऋ	ृ	.R	ॠ	ॄ
.l	ऌ	ॢ	.L	ॡ	ॣ	e	ए	े	ai	ऐ	ै
o	ओ	ो	au	औ	ौ	aM	अं	ं	aH	अः	ः

Consonants											
ka	क	kha	ख	ga	ग	gha	घ	"na	ङ
ca	च	cha	छ	ja	ज	jha	झ	~na	ञ
.ta	ट	.tha	ठ	.da	ड	.dha	ढ	.na	ण
ta	त	tha	थ	da	द	dha	ध	na	न
pa	प	pha	फ	ba	ब	bha	भ	ma	म

Semi-Vowels											
ya	य	ra	र	la	ल	va	व

Sibilants											
"sa	श	.sa	ष	sa	स

Aspirate											
ha	ह

Supplementary Consonants											
qa	क़	.kh, .K	ख़	.ga	ग़	za	ज़	Ra	ड़	Rha	ढ़	fa	फ़	La	ळ

Numerals											
0	०	1	१	2	२	3	३	4	४
5	५	6	६	7	७	8	८	9	९

Special Characters															
.o	ॐ	*AUM*	.m, M	ं	*anusvāra*	/	ँ	*candrabindu*	.h, H	ः	*visarga*
.a	ऽ	*avagraha*	@	॰	continuation	*	॑	elliptical dot	~r	�theta	Marathi *ra*
~a	ॅ	English a	~o	ऑ	English o	|	।	*daṇḍā*	..	॥	period

Table 1: The Velthuis Transliteration Scheme

Revisiting WYSIWYG Paradigms for Authoring LaTeX

David Kastrup
Kriemhildstr. 15
44793 Bochum
Germany
David.Kastrup@t-online.de

Abstract

While the command-driven approach of TeX/LaTeX has shown its power and flexibility for a variety of purposes, the lack of immediate visual feedback often renders the authoring and reviewing process itself somewhat inconvenient for both beginners and experienced users. The idiosyncratic TeX syntax does not lend itself readily to proofreading and sustained composition where the input syntax differs considerably from the produced results.

A number of approaches trying to deal with this frequently perceived shortcoming will be illustrated. On the input manipulation side there are tools ranging from a token-based approach (Syntax Highlighting, X-Symbol) to complete editors using LaTeX mostly as a means of exporting documents (TeX_{MACS}, LyX). A different range of tools does not aim to provide differences in editing, but fast and convenient access to the output results, mostly in the form of a separate page-oriented preview (Whizzy-TeX, Instant Preview). While the performance of those implementations by now leaves little to be desired, a demand for a tighter coupling of source and preview for editing purposes remains. 'Source specials' provide one way for facilitating cross-navigation between source and previews. The preview-latex package (by the author) provides a much closer coupling by directly placing previews of small elements into the source buffer.

The Moving Finger writes; and, having writ,
Moves on: nor all thy Piety nor Wit
Shall lure it back to cancel half a Line,
Nor all thy Tears wash out a Word of it.

Omar Khayyám

The LaTeX/WYSIWYG clash

What is WYSIWYG? WYSIWYG, an acronym for "what you see is what you get", is really a marketing term applied to several different degrees of similarity between the input window and the typeset output from running a system.

In its strictest sense, it means that the typeset output will be identical to the screen display. Of course, this goal is ultimately impossible: print devices have different characteristics than the screens we are working on: different pixel resolutions, different gradation of colors and gray levels. A screen dump from a WYSIWYG type word processor will typically look awful compared to a regular printout.

So what are the things people have come to expect from the WYSIWYG moniker?

Similarity to print: The editing window is supposed to resemble the printed output.

Letter shapes: While pixel accuracy is not to be expected, one might at least be able to see the general shape of the letter. Display devices often offer considerably finer amounts of control for pixel intensity than printing devices do; this makes antialiasing feasible. Antialiasing tries to compensate for a lower spatial resolution by varying the brightness of pixels according to the amount of ink that would cover the pixel given higher resolutions. Antialiased letters tend to be more readable than their non-antialiased variants, but look rather blurry as the lack of spatial resolution itself is not overcome, only its impact on the local grayness level reduced.

Extended character sets: TeX and LaTeX itself typically use an input representation based on ASCII. While there are extensions in order to be able to access 8-bit character sets, most mathematical special characters and operators such as $\sum$ and $\int$ are entered as control sequences such as `\sum` and `\int`. WYSIWYG systems usually

Figure 1: Antialiased TEX font.

Figure 2: Typical screen font.

offer a more readable rendition of such characters, as well as convenient ways to enter them without knowing their names.

Non-text elements: The most important items to mention here would be mathematical formulas and tables. Those are distinguished by conveying information more directly than possible textual counterparts.

Line breaks: WYSIWYG word processors typically have the same line breaks in the printed output as they have on the screen.

Pagination: Page breaks and figure positioning are also common elements that WYSIWYG processors will apply to the edited source in the same manner as the resulting output.

So where's the clash? It turns out that some of those targets are not the best idea for screen-based editing and document creation in general, and some are at particular odds with the way TEX works:

Similarity to print: TEX is basically a programming language. Quite often complicated instructions lead to the final typesetting results. Editing those in a WYSIWYG manner is hardly possible. WYSIWYG tends to hide abominations in the input: while text processors such as Word offer various possibilities to structure your input (format templates, text styles and so on), this is not immediately apparent in the output. As a result, average users of such systems sometimes employ rather abominable means for the formatting of their texts. Not uncommon is the use of excessive amounts of single spaces for indentation and spacings, and hand adjustments that are not robust against changes of fonts and/or printer. TEX provides an easy way of introducing comments into the typesetting source (% introduces comment lines). Such comments have no good place in a true WYSIWYG display.

Letter Shapes: In connection with TEX, the most commonly employed fonts are the Computer Modern family by Knuth. Characteristic for these fonts is a delicate balance between various stem widths and hairlines that produces a 'closed' look of the letter shapes itself while still preserving the 'leading' characteristic of serifed letters in the overall grayness level. A typical example is lower-case "t" which has a closed bowl when viewed closely, but a distribution of stem widths that effectively makes the visual impact of the closing hairline diminish (compare figure 1 to the letters in the text of this document).

Screen resolution is inadequate to properly reflect those visual characteristics. For best legibility, fonts designed for computer screen usage are to be preferred.

Extended character sets: These can lead to an easily implementable improvement in legibility. While the application is straightforward, the benefits are limited.

Non-text elements: These benefit the most from WYSIWYG representations. TEX offers considerable power for mathematics, and many extension packages make use of its macro programming features in order to gain additional functionality. For that reason, adequate rendering of compositions like math formulas and tables necessitate considerable programming efforts in the editor in order to support them well. An appropriate display of those elements is a significant aid for developing a stream of thought, and for copy editing.

Line breaks and Pagination: TEX finds its line breaks by paragraph-global optimization, and employs a process of somewhat localized optimization for finding its page breaks. An insertion mechanism caters for determining the proper amounts of additional material (such as footnotes and figures) to attach to a page. All decisions are governed by the evaluation of various kinds of penalties, and several criteria spanning more than one line are employed (visual compatibility of spacing in subsequent lines, extra penalties for adjacent lines ending with hyphenated words and so on).

The non-local layout optimization of TEX is important to achieve the best typeset results. Maintaining this during text entry itself initially appeared infeasible primarily because of performance considerations (see later for examples where this has been mostly overcome). It also can be distracting: the resulting repeated extensive on-screen text rearrangements resulting from small changes in the source make it hard to keep focus on the actual editing location.

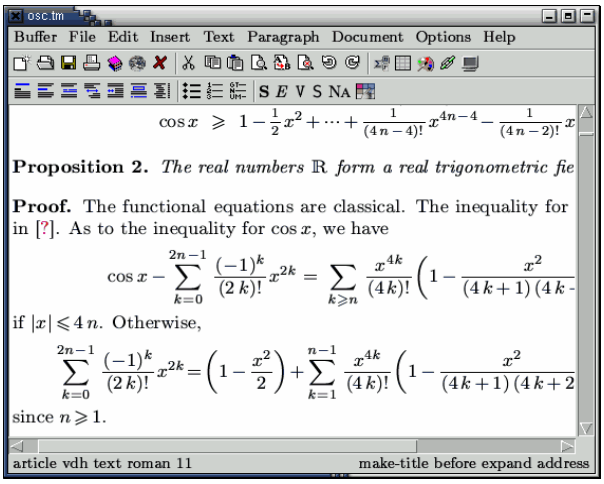


Figure 3: Screenshot from TEXMACS window.

Various systems bridging the gap

TEXmacs The word processor TEXMACS is about the clearest demonstration about what a full implementation of the WYSIWYG paradigm could look like for TEX. While TEXMACS does not make use of TEX itself nor offers convenient access to TEX packages and programming, it employs TEX typesetting algorithms and fonts (antialiased on screen) for its operation. The printed rendition of the pages is quite the same as the screen representation (though having your editing window paginated is optional). It does not support LATEX as a native format, but exports to it (needing a special style file) and, considerably less reliably, also imports from it. Its keybindings are reminiscent of Emacs' keybindings, and it will interpret quite a few macro sequences introduced with backslash. The editor can be extended and customized in Guile, the GNU project's version of the Lisp-like Scheme language.

For document versions exported to LATEX, it is possible to include direct code passages that are passed on verbatim, and that may differ from code that is used when the document is printed natively.

On a 200 Mhz system, TEXMACS appeared to respond somewhat sluggishly. While most people used to LATEX may get reasonably well along with TEXMACS, accessing the power of LATEX and document exchange with other LATEX users will be somewhat problematic.

Personally, I have found the 'concertina effect' distracting: constant reformatting during text entry causes the line spacing to shrink until the line gets wrapped differently again. While the immediate feedback from keystroke to final output is beneficial for administrating the final touches to a document, the constant fervor with which paragraphs get adjusted and shifted while single letters are being typed is about as useful and convenient as constantly busy window cleaners on a construction site.

LyX Similar to TEXMACS, LyX is a complete word processor. In contrast to TEXMACS, however, LyX promotes the WYSIWYM (What You See is What You Mean) buzzphrase instead of WYSIWYG. In earlier versions of LyX, this was mostly a euphemism for 'what you see is somewhat reminiscent of what you will get', but in more recent versions a lot of work has been invested to make the display indeed show additional information about the underlying structure of the document. Since LyX is not bound to have the input's appearance match the output, it can generously apply colors for outlining structural details, and use push buttons and similar graphic elements impacting the editing window layout for indicating footnotes and cross references.

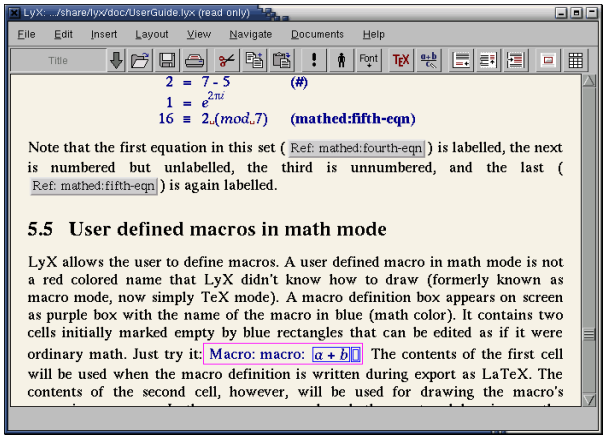


Figure 4: Typical LyX display.

While font changes and the like are reflected in the screen font selection of LyX, screen and print fonts are not the same. The screen fonts are configurable as part of the user interface, the print fonts are part of the document. It must be noted that LyX's screen display, particularly with regard to mathematics, does not seem to be optimized for readability. LyX uses LATEX for its print typesetting, employing ordinarily available class and style files. Naturally, LATEX export is unproblematic; importing it, however, is not without its difficulties. LyX uses its own formats for saving files; as with TEXMACS, sustained document interchange with other authors using pure LATEX is infeasible.

It is possible to embed LATEX code into documents even where LyX does not cater for it specifically: when LyX is unable to convert LATEX phrases

into its own format, it retains them as "Evil Red Text" (ERT) which is passed without modification into the exported LaTeX code. It is a declared goal of the LyX developers to eventually obliterate most of ERT by letting LyX natively support more and more LaTeX constructs. Considering the extent covered by LaTeX (a lot of which does not seriously gain usability by a separate screen representation), this means a continuing drain of resources. In particular, things like LaTeX3 will need much work to accommodate.

One of the strengths of LyX is its math editor, quite similar to that of TeXmacs. LyX has the same update-per-keystroke policy that, in connection with justification, leads to the concertina effect of shrinking and expanding lines during normal text insertion.

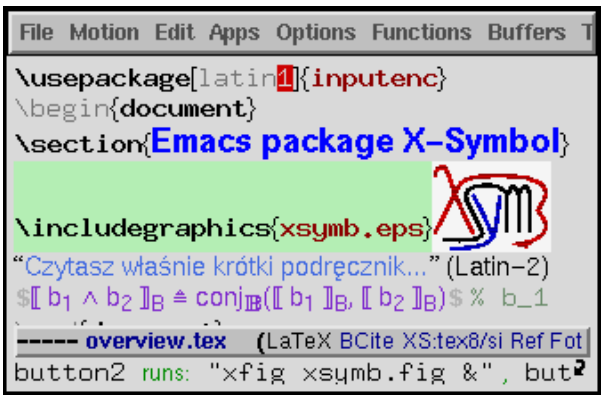


Figure 5: X-Symbol at large.

X-Symbol X-Symbol is the first system presented here falling short of providing a complete text processing system 'merely' for the sake of introducing WYSIWYG into TeX authoring. It is not an accident that, like all solutions presented here not coming with their own editor, it runs under the extensible editor Emacs (or its variant XEmacs). Emacs is

- intended for editing source texts
- free software
- a powerful editing environment
- easily extensible with a highly interactive extension language (Emacs Lisp)
- available for a large number of different operating systems

Since TeX is a multi-platform free software system with an editable plain text source language, Emacs and TeX are matched rather well. The most commonly employed package for Emacs providing an extensive editing and runtime environment for TeX and LaTeX is called AUC TeX.

So what does X-Symbol do? It pretty much exclusively deals with the 'extended character set' angle of WYSIWYG. It will display the likes of `\sum` and `\leq` as "$\sum$" and "$\leq$" and will also cater for accented letters in text, such as `\"a` being displayed as "ä". Its operation is limited to unique short control sequences. Some of the employed fonts come with X-Symbol and fit in with the normal monospaced screen fonts of Emacs. This means, for example, that operators like $\int$ will look rather small on-screen. As an added convenience, it has special fonts that it can use in connection with syntax highlighting to make super- and subscripts appear with smaller letters and appropriately displaced. While it will automatically convert control sequences into its special characters as soon as they are typed, it also offers a few other methods of directly accessing those characters.

X-Symbol actually replaces the characters in question in the source text buffer, and just converts them back into their TeX equivalents when writing out files. There are several potential problems which can lead to files being permanently changed:

- If for some bug the text does not get converted to TeX readable form when written out, you'll end up with basically illegible garbage. Some effort is needed to get control sequences back.
- The combination of reading file in under control of X-Symbol and writing it out again is not guaranteed to be unique. This is particularly a problem in verbatim-like settings. Recent versions of X-Symbol behave much more predictably and cautiously than previous ones, however.

Since the internal presentation of the text is changed to extended characters, this means that searching for those control sequences with the usual search functions of Emacs becomes awkward to infeasible.

One part of the value that X-Symbol provides to the user is an editor-level replacement for LaTeX's inputenc package (for plain TeX, this could be even more important). Using X-Symbol is a particularly convenient option when your text would otherwise mandate switching between input encodings in a single document. In that case, a coherent editor display would be hard to achieve. With X-Symbol, the characters from the 'foreign' encoding are expressed in appropriate control sequences when saving, obliterating this particular problem.

But the most important addition of X-Symbol is probably the support for numerous ways of inputting the characters it caters for, ranging from keyboard shortcuts to the 'grid' (a large menu with

all of the available characters displayed in an editing buffer) and entry via the menubar. Apart from the grid, those techniques are not really WYSIWYG-related.

X-Symbol is mainly an input prettifier, converter and accelerator: it does not provide for any previewing facilities. It also does not cater for any more complicated compositions, like formulas. In that respect it does a lot less than systems like LyX. On the other hand, it provides no obstacles or inconveniences with regard to accessing the full power of LaTeX, and it will work also with SGML or HTML as well as plain TeX.

See the illustration in figure 6(b) for a more typical illustration of the visual buffer effect, this time in connection with preview-latex.

preview-latex preview-latex also is an Emacs add-on package. In contrast to X-Symbol, it is concerned exclusively with the display aspect of WYSIWYG. It does not change the text of the edited buffers at all, and has no impact on the format of external files. While the aim of X-Symbol is to replace single control sequences with letters and symbols matched to the screen fonts, preview-latex interchanges the source text display of whole compositions (formulas, section headers, figures, included graphics) with a proper antialiased preview obtained by running the LaTeX passage in question through LaTeX, Dvips and Ghostscript. A single call of LaTeX (potentially with a pre-dumped format) can provide previews for the entire document, and Ghostscript will be used as a rendering daemon working from a single PostScript file, processing those images currently on-screen with priority. Future versions will mostly bypass Dvips and Ghostscript, but update speed is already quite workable even on modest hardware. Another future option will be to replace LaTeX and Dvips with PDFLaTeX.

The previews are made to match the default screen font in background and foreground color, as well as in scale. Other than that, they are identical to actual print previews. Since preview-latex does not know about compositions' inner structure, it will just provide mundane source text display while you are editing them, switching back into graphical preview mode when you indicate you are finished.

What elements in a text are actually considered previewable compositions is determined by an external LaTeX style. Its operation can easily be customized by placing declarative commands in the document preamble or a separate configuration file.

The underpinnings of preview-latex are not particular to the Emacs editor: similar functionality is being implemented for LyX, currently restricted to its math mode.

Whizzy-TeX Whizzy-TeX is one of a number of systems that focus on automatic fast updating of a print preview in a separate window. It is (surprise, surprise) an Emacs package and best complemented with the previewer Active-DVI (written in Objective CAML) from the same author, since that previewer can switch to the correct page and location without flashing when the DVI file changes. While you can use Whizzy-TeX also with XDvi, the update action is less smooth.

Whizzy-TeX is a preview system that continually tracks cursor movements and text changes, and in case of a change, reruns LaTeX from a recent point where it has made LaTeX dump its state into a format file. That way, the DVI updates occur quite fast, and it becomes feasible to play around with stuff influencing typesetting decisions.

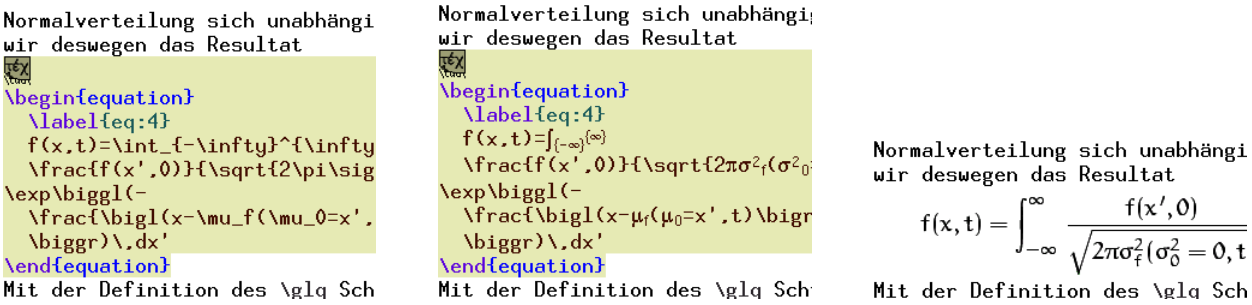
Take a look at subfigure 7(e) for an illustration of Whizzy-TeX in connection with Active-DVI.

ActiveTeX/Instant Preview ActiveTeX's core is a constantly running TeX process called the TeX daemon, which typesets pages on demand. A separate program then processes individual pages from the resulting DVI file as they get produced.

The Instant Preview package for Emacs will use this for keystroke level updates of a TeX buffer. Since TeX does not get restarted, the material processed in this manner should mostly be stateless so that repetitive runs work well. For that reason, the system is mostly unsuitable for LaTeX. The principal author of the system uses XDvi for the display; it might well be that Whizzy-TeX's Active-DVI could provide a smoother update action and avoid flicker.

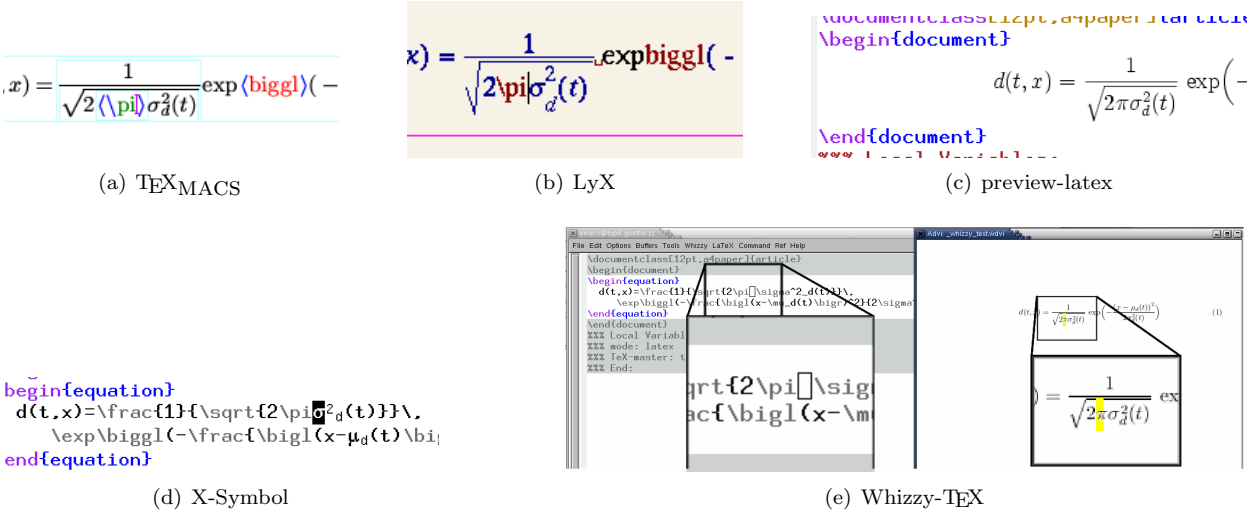
The main aspect of ActiveTeX is raw speed on low hardware. Apart from that, it offers little if any advantage over Whizzy-TeX and serves similar goals, while being less well-suited for LaTeX.

Source Specials Source specials are an editor/previewer coupling tool that works by placing special marks into the produced dvi file that indicate the source location where the dvi file results originated from. These can be either inserted with a special LaTeX style, or automatically by most newer TeX implementations. In combination with support in both editor and previewer, one can implement forward search (the position in the editor gets automatically tracked in the preview window) and reverse search (clicking into the preview window will relocate the cursor in the editor window to the corresponding source line). This is exclusively a cross-navigational tool. It serves no actual WYSIWYG



(a) Opened equation (b) The same, with X-Symbol (c) The same, closed

Figure 6: Using preview-latex with and without X-Symbol.



(a) TEXMACS (b) LyX (c) preview-latex

(d) X-Symbol (e) Whizzy-TEX

Figure 7: Detail views.

functionality. We mention it here because it illustrates the perceived need for a closer coupling between editing window and preview.

Summary

Figure 7 shows screen shots of how the various systems treat a formula in display, and an overall summary can be found in table 1. Those WYSIWYG systems that try to provide a more customary input experience suffer from the handicaps of

- having to implement a complete editing environment on their own.
- needing to be able to interpret all of the supported constructs by themselves instead of letting LaTeX do the work. In that way, only a selected subset of LaTeX can be supported properly and efficiently, and the developers of both TEXMACS and LyX have chosen to employ a native format different from LaTeX for working purposes, which makes accessing LaTeX as an external interchange format largely unfeasible. Take a look at figure 7 for how the unknown delimiter size specifiers are treated by those first two systems.
- Needing to always show a coherent editing display during operation. This makes it infeasible to actually generate the display with LaTeX.

Systems with keystroke level reformatting and justification in the input window some users find distracting for continuous text entry. This sort of operation costs considerable performance on slower systems, and it renders the canvas unquiet. Repercussions to larger areas are counterproductive in creation mode, while desirable for admistering final touches to a document. While Whizzy-TEX also does keystroke update, it does so in a separate area which is less of a distraction but has the disadvantage of requiring a different focus of attention in case you actually need to look at the typeset contexts. preview-latex economizes updates (only done

Table 1: Ratings of available tools.

	TeX$_{\text{MACS}}$	LyX	X-Symbol	preview-latex	Whizzy-TeX
Portability	−	−	−	0	+
Source preservation	−	−	+	++	++
Keystroke updates	yes	yes	yes	no	yes
Uses editing window	yes	yes	yes	yes	no
Edit responsiveness	−	0	+	N/A	N/A
Edit accuracy	++	0	−	+	N/A
Preview speed	++	0	N/A	+	+
Edit screen estate	+	+	+	+	−
Preview estate	+	−	N/A	+	−
Cross-navigation	++	−	−	+	0

Description of categories:

Portability	How easy will it be to transfer this system's mode of operation to other editors and editing platforms?
Source preservation	How faithfully will existing LaTeX document source be preserved when editing?
Keystroke updates	Does the system perform its updates on a per keystroke level?
Uses editing window	Does the system work within the editing window?
Edit reponsiveness	How fast does the system process keystrokes while editing? N/A if the system has no influence on keystroke processing time.
Edit accuracy	How close to the typeset result is the representation in the editing window?
Preview speed	How fast will a true preview be available?
Edit screen estate	During normal operation of the system, how much screen real estate is needed?
Preview estate	If in need of a true preview, how much screen estate will be needed? N/A for packages which don't have a default way of previewing.
Cross-Navigation	How easily can we establish the correlation between a true preview and the corresponding source position?

Description of ratings:

++	very good
+	good
0	fair
−	unfavorable
N/A	not applicable

on request, which is very easy), screen real estate and focus, at the cost of not providing any preview of the current object you are editing. This can be improved somewhat by employing X-Symbol which offers additional input convenience.

The upcoming combination of LyX's math editor with preview-latex-like functionality is an interesting development. While experienced LaTeX users might prefer linear text entry, the combination of easy access to both readably composed input as well as perfectly typeset LaTeX code will be a great help for advanced users exploring beyond the LaTeX constructs still fully featured by the LyX editor.

On the input side of WYSIWYG, a reasonable compromise under Emacs is provided by X-Symbol. On the preview side, once you leave the specialized editor area (where TeX$_{\text{MACS}}$ offers the most integrated approach), the preview-latex paradigm appears most useful for general implementation in editing systems: use of LaTeX for the typesetting ensures high accuracy while yielding full and unencumbered access to the full power available from LaTeX. For the kind of syntactical units that preview-latex processes, the lack of a per-keystroke update policy (unique among the presented tools) is in practice an advantage since it allows the user to compose the unit without distraction and commit it only when it is actually ready to be run through LaTeX. For interactive changes in connection with adjusting page layout material, Whizzy-TeX provides the user with

fast updates. The added screen estate, and the separate preview area render it less optimal for other tasks in copy editing and document creation. That Whizzy-TEX has two competing screen locations of interest becomes apparent in figure 7(e): no other screen shot required the use of magnification glasses.

An advantage of Whizzy-TEX is that it provides a preview framework which can easily be embedded into different editors: it should be conceivable to adapt it to provide an alternative previewer for LyX, for example.

Future developments

The complete text editing environments will continue to support more LATEX constructs. Performance increases will not have much of a further impact on the usability of currently available approaches. The most interesting approaches to watch may be hybrid ones which may make their way into non-Emacs based editing solutions eventually. As an example, LyX will come with functionality similar to preview-latex in its next major version (presumably 1.3.0), at first just for previewing math (due to technical reasons).

Prospective embeddable TEX components (Ωlibraries, DVI-rendering daemons) might make impact on the operation of display engines. The ligaturing and compositing mechanisms that constitute TEX's backend might make a good object for integration into existing word processors in a manner similar to TEX$_{\mathrm{MACS}}$. Perhaps drop-in equation creation components based on TEX code might become more prevalent in free software systems eventually.

Availability

Here is where the packages can be found on the Internet (the leading `http://` has been omitted):

TEX$_{\mathrm{MACS}}$	`www.texmacs.org`
LyX	`www.lyx.org`
X-Symbol	`x-symbol.sourceforge.net`
preview-latex	`preview-latex.sourceforge.net`
Whizzy-TEX	`pauillac.inria.fr/whizzytex`
Active-DVI	`pauillac.inria.fr/advi`
ActiveTEX	`www.activetex.org`
Emacs	`www.gnu.org/software/emacs`
XEmacs	`www.xemacs.org`
Src Specials	`xdvi.sourceforge.net/inverse-search.html`
Ω	`omega.cse.unsw.edu.au`

All of the described packages are released under the GNU General Public License and are thus free software (LyX is released under a modified version in order to allow linking with the XForms library).

serendiPDF with Searchable Math-fields in PDF Documents

Ross Moore
Mathematics Department, Macquarie University, Sydney
ross@maths.mq.edu.au
http://www.maths.mq.edu.au/~ross/

Abstract

serendiPDF is an attempt to make it easier to find the correct way to express complicated mathematics, especially aligned environments, using LaTeX. This is achieved by storing a copy of the LaTeX source for a mathematical environment inside the generated PDF output, in a way that allows it to be easily accessed and copied into the source for other documents. In this way, the full power of "serendipity", as a means for appreciating and learning unfamiliar techniques, becomes available for authors of mathematical LaTeX documents.

The existence of extra (initially hidden) mathematical fields within PDF documents, allows for a solution of the perennial problem of how to search for pieces of mathematics within typeset documents. A solution is presented whereby symbol names, such as `\alpha` (α), `\Gamma` (Γ) and `\Sigma` (Σ), can be located within the extra math fields. The interface behaves just as one would expect from a search-engine, finding fields either anywhere within the document, or limiting the search to just the currently visible page.

Serendipity

Dictionaries define serendipity as the act of "accidental discovery", such as finding something of value by accident, when actually looking for something else. In the context of modern computing software, with extensive menus and an intricate graphical interface, serendipity clearly plays a rôle in learning how to use the program. When searching through the menus for the way to perform a particular kind of task, one frequently tries out unfamiliar options. In doing this one may not find what was being looked-for, but instead discover how other tasks can be performed. Typically 'features' discovered in this way are remembered, or appreciated, much better than if a manual had been consulted.

With what used to be called WYSIWYG word-processing software ("What You See Is What You Get"), serendipity can play a significant rôle in constructing complex documents. Rather than learning from a manual how to (for example) create a tabular layout or complicated mathematical expression, one just copies something that looks like it does part of what seems to be needed, then makes alterations until it is presenting what is desired. This may not lead to the best possible appearance, or the most efficient (in some sense) coding, but it can get the job done.

To TeX purists this can be anathema—data should be presented in a consistent logical manner, with appropriate mark-up to indicate its meaning, not just appearance. While this is true, it also leads to the perception that TeX (or LaTeX) is difficult, both to use effectively, and to learn—despite its obviously superior output quality. This author contends that the problem is largely due to the 3-step edit–compile–view cycle that is at the core of document preparation using TeX. An inexperienced user can see superb TeX-produced output, but may not know what kind of input source was required to produce it. For all but simple text and paragraphing, it is generally not possible to take the output and reuse it (with appropriate edits) in a new document, without having access to the author's original source coding, or similar work. At least with PDF as the output format, it is possible to capture text using the 'Text Capture' tool. But try to use this tool for mathematics or tables—it just does not help at all.

Such an edit–compile–run cycle used to be the predominant computing paradigm, so it was no surprise that TeX was constructed to work in this way. Nowadays however, many people have used computers effectively for a large number of years without ever (knowingly) having compiled a program; the concept is something completely foreign to them. This can be true of students and academics in all

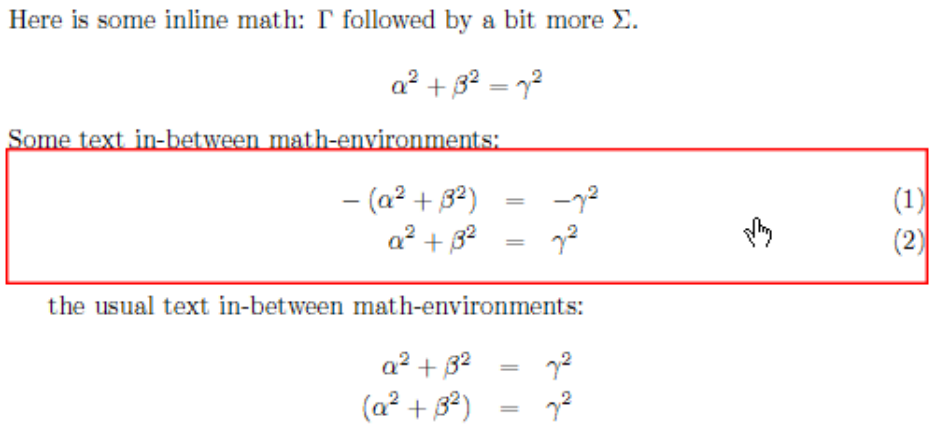


Figure 1: Moving the mouse over a piece of typeset mathematics causes the outline of an invisible button to be displayed. Clicking on this button toggles the visibility of a field with LaTeX source, see figure 2.

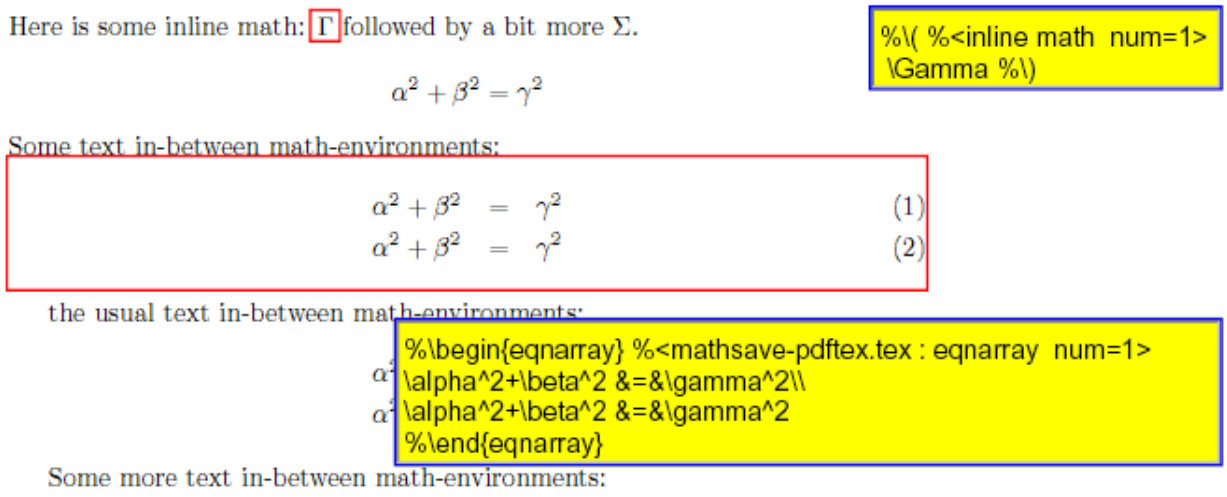


Figure 2: In response to a click over a piece of typeset mathematics the visibility of a math-field is toggled. Here we see how the field contains the LaTeX coding for the typeset mathematics, as a complete environment.

fields, so it is not hard to see why TeX has been described as "arcane", and does not occupy the prominent place in publishing that befits the quality of its output. It is the *lack of serendipity* that makes learning TeX seem to be so much harder than for other modern software packages.

The main purpose of serendiPDF is to implement an idea that may help to change this. Now mathematics *can* be recovered from specially prepared PDF documents, using nothing more than the Acrobat Reader [2] provided (free of charge) on all platforms by Adobe Systems Inc. The idea is to include the LaTeX source for mathematics environments as hidden fields within the PDF document. Visual clues indicate the presence of these fields, as indicated in figure 1. In response to a single mouse-click a field can be shown, thereby revealing the LaTeX coding which has then been 'discovered' serendipitously; see figure 2.

A LaTeX document can be prepared such that *all* mathematics coding is included also within these hidden fields. Such a document becomes not only a valuable source of scholarly information on the topic being presented, but also a useful example for learning how to create the high-quality appearance for

Figure 3: The 'Search for math' control box allows TEX source to be located within math fields. Searches normally look forward from the current page, but check-boxes allow it to cover the whole document or restrict to the one page.

mathematical information of a similar kind. Inexperienced authors can learn from the older masters, not just the intricacies of the meaning embodied in the mathematics, but also how best to present it.

Having the LATEX coding for mathematics available leads to further useful possibilities. For example, it now becomes a simple matter to search for mathematical expressions within a PDF document (see figures 3 and 4); something which was hitherto quite impossible. It should even become possible to develop plug-in software that allows mathematical expressions to be edited in-place.

The serendip package, with insdljs.sty and hyperref.sty

Electronic fill-in forms are have now become quite common, in both HTML and PDF. For the most-part, these rely upon a JavaScript interpreter being available within the web-browser or PDF reader software. (JavaScript is a programming language that handles the appearance of buttons and the showing/hiding of fields and annotations, as well as calculations, and a myriad of other kinds of computing task related to a document and its content.)

Acrobat Reader [2] has had JavaScript support since version 4.0 [1], and is even more sophisticated in versions 5.0 and later. Donald Story [7] has been pioneering the use of JavaScript within PDF documents generated from TEX source, for several years. His insdljs package (acronym for 'Insert Document-Level JavaScript') provides coding that allows functions and procedures to be included within PDF documents, using any of dvips (+ Ghostscript or Distiller) or dvipdfm or pdfTEX as the engine producing the final PDF output. In fact with pdfTEX [4] the inclusion of JavaScript is relatively straight-forward, using just the hyperref package to help specify fields and buttons. With other drivers, the pdfmark [3] technique is used extensively.

For placing mathematics code into fields, the serendip package builds upon the support for fields, buttons and JavaScript in both the insdljs and the hyperref packages. The serendip package works with LATEX math environments, such as `\(....\)` for in-line, and `\[....\]` for displayed math, as well as the equation, eqnarray and displaymath environments. It also works with the outer-level environments and alignments defined within the amsmath package. It does this by redefining, in a non-destructive way, the behaviour of some LATEX and AMS macros.

For example, with the equation environment, as started by `\begin{equation}`, it is the macro `\equation` which is redefined to behave as follows.

1. Open a new level of grouping and redefine the `\catcode`s of non-alphabetic characters.
2. Read to the corresponding `\end{equation}`, to get the complete source for this environment.
3. Store this LATEX source as a list of tokens, to be later written as plain text into a hidden field within the PDF document being constructed.
4. Estimate the size (both height and width) required for the text field by typesetting the tokens in a `\hbox` using a fixed-width font then measuring the result.
5. Write the list of tokens into a file (with `.mth` extension) so it can be re-read by TEX with the usual `\catcode` values.
6. Construct the text-field containing the LATEX source, named sequentially with all equation environments. and positioned using `\marginpar`.
7. Close the grouping level, reverting `\catcode`s to their normal TEX values.
8. Start a new inner `\vbox`, to hold the typeset mathematics.
9. Read-in the contents of the `.mth` file for typesetting, using a stored pointer to the usual expansion of `\equation`.
10. Measure the size of the resulting `\vbox` and construct a button of this same size.
11. Place the `\vbox` onto the page with the correct amounts of preceding and trailing glue.
12. Remove trailing glue, remembering how much was used.
13. Place the button also onto the page, directly over the typeset mathematics. This button will be used in the final PDF to toggle visibility of the text-field.
14. Replace the trailing glue, so that the environment interacts correctly with material following afterwards.

The resulting page may differ slightly in the stretchability of the glue around the environment. Mostly this is not noticeable at all. Other environments are handled similarly, except for inline mathematics, where the typesetting is done within an `\hbox`, which is measured and later placed onto the page. Thus inline-math environments must occur entirely

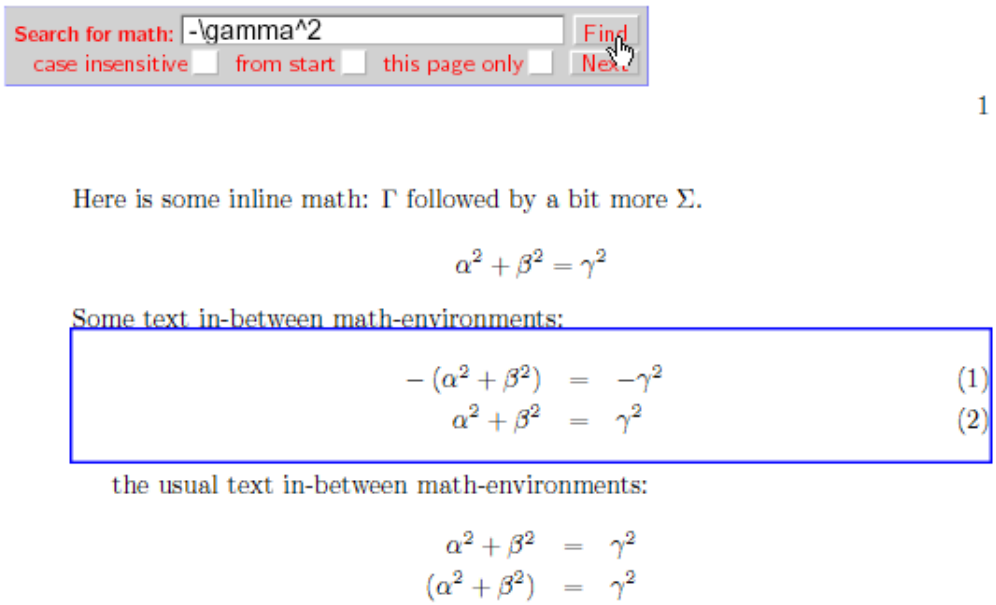


Figure 4: When the specified math coding has been found, the environment is indicated by highlighting the border of its overlying button, and giving focus to that button. The field itself is not shown until the button is pressed. Repeated use of 'Next', instead of 'Find', cycles through subsequent occurrences.

within a single line; any breaks need to be forced, resulting in two fields and a button for each.

Searchable Mathematics

One consequence of having the LaTeX code for mathematics available in text-fields within the PDF document, is that now it becomes possible to search these fields using JavaScript procedures. The mathsrch package constructs a console, as shown in figure 3. This allows mathematical expressions to be found by searching forward within the math-fields embedded within the current PDF document.

By checking a box, a search can be modified to restrict to just the current page, or to cover the whole document, starting from the beginning. Using the 'Next' button, rather than 'Find', allows all occurrences of a particular expression can be located sequentially. To indicate a found expression, the mathematical environment is indicated by outlining its button, as shown in figure 4. If the LaTeX code itself needs to be shown, then an extra click is required on this outlined button.

Placing the search-console is not an automatic consequence of loading the mathsrch package. Certainly this constructs the console as the contents of a TeX box register, called `\MathSearchBox`. This box can be placed anywhere on a LaTeX page, using the command `\MathSearch`, which is just a macro that expands to `\copy\MathSearchBox`. (It is important to use `\copy`, rather than `\box`, so that that console can be used repeatedly on different pages.)

Since space characters are usually ignored in (La)TeX math-mode source, some flexibility is built into the searching mechanism. A space token in the search-string is not required to match in the math-fields; in fact, it can match any number of spaces, including none at all. For example, `x+y` matches only `x+y` in a math-field; but `x + y` will find any of `x+y`, `x +y`, `x+ y`, `x + y`, as well as `x +y` and `x + y`, and other strings having more spaces.

The mathsrch package also defines a command `\MathSearchInHeader`, which can put a console on every page, situated neatly above the header and abutting into the left-hand margin. More precisely, `\MathSearchInHeader` calls upon another macro, `\PlaceMathSearchBoxInHeader`, which expands as:

```
\newcommand{\PlaceMathSearchBoxInHeader}{%
  \pagestyle{myheadings}%
  \markboth{\protect\MathSearch\hfill}%
   {\protect\MathSearch\hfill}}
```

From this it can be seen that the `\pagestyle` is set to be myheadings, and the header contents are given explicitly. If other page-styles are being used, then it is appropriate to make a re-definition:

```
\renewcommand{\PlaceMathSearchBoxInHeader}{%
  \pagestyle{.....}%
  \markboth{.....}{.....}}
```

to accommodate the desired page-style and header.

Customisation of the search-panel is also possible; e.g., for a language other than English, or to change width and colours. A file msearch.cfg is read, if it can be found on any of the usual LaTeX search-paths. The customisable para meters are provided with default values as follows:

```
\providecommand{\msearchString}{Search for math:}
\providecommand{\msearchFindString}{ Find }
\providecommand{\msearchNextString}{ Next }
\providecommand{\msearchPageString}{this page only}
\providecommand{\msearchCaseString}{case insensitive}
\providecommand{\msearchStartString}{from start}
\providecommand{\msearchText}{put TeX code here}
\providecommand{\msearchWidth}{3.25in}
\providecommand{\msearchFGcolor}{red}
\definecolor{ltgray}{gray}{.85}
\providecommand{\msearchBGcolor}{ltgray}
\providecommand{\msearchBorderColor}{blue}
\providecommand{\msearchBorderOpts}{borderstyle=B,
  borderwidth=1, bordercolor= .85 .85 .85}
```

Any of the above macros can be given different expansions either within mathsrch.cfg, or *before* the mathsrch package is loaded, or values changed *after* the package has been read, using `\renewcommand`.

Future Developments

The original intention for having math-fields is to allow the less-obvious parts of a document's TeX source to be distributed along with the final PDF. This is meant primarily as a teaching aid. It remains to be seen whether it will indeed be used in this way; or if other applications are found, once this extra enrichment of PDF documents becomes more widespread.

For example, the code from a math-field can be edited in-place, and subsequently used to generate a modified form of the mathematical environment. This can be done in an external program or utility, such as the 'Equation Service' by Bob Rowlands [6] for Macintosh OS X, that works as a process callable from other running applications. (In effect, pdfTeX generates a new image of just the modified mathematics.) It should be possible to use the plug-in technology for Adobe's Acrobat (full version, not just the Reader), to include such an image into the original PDF, for display in the same location as the typeset mathematics from which it was derived. By merging this image as an update to the original document, we would have what is effectively PDF editing capabilities for touching-up TeX-typeset mathematics.

Acknowledgements The author wishes to thank Donald Story, for many email discussions and help with suggestions about how to implement various technical ideas using JavaScript. Without his work on insdljs and the exerquiz packages, and testing of example documents, the serendip and mathsrch packages described here would have taken much longer to complete.

References

[1] Adobe Systems Inc.; "Acrobat Forms JavaScript Object Specification, Version 4.0"; Technical Note #5186; Revised: January 27, 1999.

[2] Adobe Systems Inc.; Acrobat Reader, viewer for PDF format documents, available free of charge from `http://www.adobe.com/`.

[3] Adobe Systems Inc.; "`pdfmark` Reference Manual"; Technical Note #5150; Adobe Developer Relations; Revised: March 4, 1999.

[4] Hàn, Thế Thành; pdfTeX, free software for generating documents in PDF format, based on the TeX typesetting system. Available for all computing platforms; see `http://www.tug.org/applications/pdftex/`.

[5] Netscape Communications Corporation; Netcape JavaScript Reference, 1997; online at `http://developer.netscape.com/docs/manuals/communicator/jsref/toc.htm`.

[6] Rowland, Bob; 'Equation Service', program for Macintosh OS X to produce small PDF images of TeX-typeset mathematics or text; version 0.5b, 2002. Software available online from `http://www.esm.psu.edu/mac-tex/EquationService/`.

[7] Story, Donald; `exerquiz` & AcroTeX, packages for including special effects in PDF documents, using TeX and LaTeX. Dept. of Mathematics and Computer Science, University of Akron. Software available online from `http://www.math.uakron.edu/~dpstory/webeq.html`.

[8] Story, Donald; "Techniques of Introducing Document-level JavaScript into a PDF file from a LaTeX Source". TUG 2001, TeX Users Group Annual Meeting, Delaware, August 2001.

A Conversion of Public Indic Fonts from METAFONT into Type 1 Format with TEXTRACE

Karel Píška
Institute of Physics, Academy of Sciences
182 21 Prague, Czech Republic
piska@fzu.cz
http://hp18.fzu.cz/~piska/

Abstract

The paper presents fonts for Indic languages in the Type 1 format converted from METAFONT sources with the TEXTRACE program, developed and presented by **Péter Szabó** in 2001. TEX supports major Indic scripts and the TEX/LATEX packages together with public font METAFONT sources are available in the TEX archives (CTAN) in the `tex-archive/language/<lang>`. The fonts in the `pfb` format, despite their limited quality of approximation and relatively large font file size, may be used as an alternative to corresponding bitmap fonts (represented by `pk` files), permitting creation of documents in PDF containing only vector outline fonts, and eliminating the use of bitmap fonts.

Introduction

Outline fonts are preferable and more suitable for use in PDF or other final electronic documents than bitmap fonts. Among the outline fonts, the most popular in the TEX world are Adobe Type 1 PostScript fonts [2]. A single definition of a Type 1 font embedded in a document may cover all the magnifications of the script, is common for all resolutions, and the result can be zoomed in a previewer (like Acrobat Reader), scaled in PostScript printers and devices, etc. It is not necessary again and again to generate bitmap font representations for various sizes and different resolutions of output devices from METAFONT sources. Moreover, the bitmap images look ugly and are displayed slowly.

Today most of the major Indic scripts are available for use with TEX. Numerous fonts in the METAFONT format together with TEX support are available from the Comprehensive TEX Archive Network (CTAN). "An Overview of Indic Fonts for TEX" was published by Anshuman Pandey in *TUGboat* in 1998 [1]. Following the example of this article we will use the adjective "Indic" (not "Indian") for languages and scripts of India. Several public Indic fonts in the PostScript Type 1 form exist in CTAN but the exact Type 1 equivalents of the METAFONT originals usually are not freely available. For TUG 2002 in India I decided to prepare a collection of Type 1 Indic fonts corresponding to their METAFONT sources from CTAN.

The TEXTRACE program developed by Péter Szabó [3] was used for conversion into the Type 1 format from the original METAFONT sources. The font families, converted and used in the contribution, are listed in Table 1 with their versions and author names.

Several public Indic fonts in the Type 1 format (listed in Table 2) can be found in CTAN. Other Type 1 fonts for the Indic languages exist but they are not free or are not available from CTAN. Table 3 presents an overview of the Unicode "common" part of consonants for Indic languages — now in the Type 1 representation.

Conversion with TEXTRACE and the first stage postprocessing

I intended to test possibilities of the TEXTRACE program providing a conversion of bitmap images into outlines and I did not want to "reinvent" the methods of analytic conversion developed by Basil K. Malyshev [4] or Richard J. Kinch [5] despite the loss of important information from METAFONT.

The conversion from METAFONT sources into a primary Type 1 format was executed on a SUN workstation with the Solaris 2.6 operating system without problems; no assistance was needed and it took about 10 minutes for a single font.

Unfortunately, after the TEXTRACE transformation, the Type 1 fonts, without optimization and without hinting, are not perfect; the outline curve approximation contains many nodes, the font files

are relatively large, and postprocessing is necessary. The results of the outlines approximated by TEXTRACE are worse than I had expected.

The converted font accurately reflects the original *as a whole* and successfully solves a minimalization problem. The primary Type 1 fonts, however, have a number of inconvenient features *in the details*: absence of extrema points and, on the other hand, unexpected bumps may occur; improper starting points; straight lines divided into multiple segments; identical glyph elements may be approximated differently by segments; corners with small angles are not detected and are approximated by arcs or sequences of short segments; node clustering in regions where the curvature is changing; irregularities due to "*simplification*" of some "*details*" in the METAFONT sources; (most notably) short segments have bad tangential angles; slanted straight lines are approximated by curves — and — curves are approximated by straight lines (contrary to other known font families of our experience); single curve segments in the "S" form (tangential vectors from nodes point in contrary directions).

The subsequent multistep postprocessing is and will continue to be executed using T1UTILS [6] for the conversion of `pfb` files into a readable "raw" text form and back. The AWK programs [7] are used for operations with quasi-automatic or manual marking of the "raw" form by marks — commands, e.g., for inserting extrema points (i.e., a segment is divided into two segments in a point with partial derivation equal 0); merging two curve segments (the previous subpath is changed into a more smooth segment); joining more segments into one straight line (after TEXTRACE it may be split) or by moving the starting point of a path to the next or the previous node. Hinting information (in the TEXTRACE output it is missing) may be added by a font editor using an autohinting tool, e.g., using PFAEDIT [8].

Unfortunately attempts to develop automatic postprocessing algorithms have been unsuccessful. Different fonts have different designs and "*produce*" different problems. Many irregularities are "*local*" and a more general approach is impossible to find.

Availability

The Type 1 fonts (`pfb` files) in α-version in various stages of postprocessing are or will be available from my web site (`http://hp18.fzu.cz/~piska/`) together with the corresponding `tfm` files (copied from CTAN) and proofsheets in PDF. The proofsheets are created using my method of generating associated Type 1 fonts that visualizes nodes, control points and hints. This method was used for a comparison of CM/EC fonts and was presented at the EuroBachoTEX2002 meeting (April 29–May 3) in Poland [9].

Acknowledgements

I would like to thank all the authors of public METAFONT fonts for Indic languages.

Conclusion – Future work

The outline Type 1 fonts can be relatively simply generated by TEXTRACE from METAFONT sources and then can successfully be substituted for their bitmap versions. But large sizes of font files and many "*tiny*" details on glyph outline level need complicated postprocessing.

References

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

[2] Adobe Systems Inc. *Adobe Type 1 Font Format*. Addison-Wesley Publishing Company, 1990.

[3] Péter Szabó. "Conversion of TEX fonts into Type 1 format", *Proceedings of the EuroTEX 2001 conference*, pp. 192–206, Kerkrade, the Netherlands, 23–27 September 2001.

[4] Basil K. Malyshev, "Problems of the conversion of METAFONT fonts to PostScript Type 1", *TUGboat*, **16** (1), pp. 60–68, 1995.

[5] Richard J. Kinch, "MetaFog: Converting METAFONT Shapes to Contours", *TUGboat*, **16** (3), pp. 233–243, 1995.

[6] T1UTILS package. (Type 1 tools). `http://www.lcdf.org/~eddietwo/type/#t1utils`

[7] Alfred V. Aho, Brian W. Kernighan, and Peter J. Weinberger, *The AWK Programming Language*, Addison-Wesley, 1988.

[8] George Williams. PFAedit – A PostScript Font Editor, `http://pfaedit.sourceforge.net/overview.html`.

[9] Karel Píška. "A comparison of public CM/EC fonts in Type 1 format", *Proceedings of the XIII European TEX Conference*, April 29–May 3, 2002; Bachotek, Poland.

Table 1: Indic METAFONT fonts converted into Type 1

	Script	Font Name (Package)	Author(s) of METAFONT `CTAN/language` subdirectory	Version
Dvg	Devanagari	`dvng10` (devnag)	Frans J. Velthuis `indian/itrans`	1.7 (1998)
San	Sanskrit	`skt10`	Charles Wikner `sanskrit`	2.0 (1996)
Ben	Bengali	`bnr10`	Abhijit Das `bengali/pandey`	1997
	Gujarati		No METAFONT font in `CTAN`	
Gmu	Gurmukhi	`grmk10` (Gurmukhi)	Amarjit Singh `gurmukhi/singh`	1.0 (1995)
Pun	Punjabi	`pun10`	Hardip Singh Pannu `gurmukhi/pandey`	1999
Ory	Oriya	`or10` (Oriya-TEX)	Jeroen Hellingman `oriya`	0.93 (1998)
Sin	Sinhalese	`sinha10` (sinhala_TEX)	Yannis Haralambous, Vasantha Saparamadu `sinhala`	2.1.1 (1996)
Kan	Kannada	`kan10` (KannadaTEX)	G.S. Jagadeesh `indian/itrans`	1991
Tel	Telugu	`tel10` (TeluguTEX)	Lakshmankumar Mukkavilli `telugu`	1.0 (1991)
Mlm	Malayalam	`mm10` (Malayalam-TEX)	Jeroen Hellingman `malayalam`	1.6 (1998)
Tam	Tamil	`wntml10`	Thomas Ridgeway `tamil/wntamil`	1988–91
Tib	Tibetan	`ctib` (cTibTEX)	Sam Sirlin, Oliver Corff `tibetan/ctib`	0.1 (1999)

Table 2: Indic Type 1 fonts from CTAN

Script	Font Name	Directory	Author of Type 1 font
Devanagari	`xdvng`	`indian/itrans`	Sandeep Sibal
Bengali	`ItxBengali`	`indian/itrans`	Shrikrishna Patil
Gujarati	`ItxGujarati`	`indian/itrans`	Shrikrishna Patil
Punjabi	`Punjabi`	`indian/itrans`	Hardip Singh Pannu
Perso-Arabic	`xnsh14`	`arabtex`	Taco Hoekwater

Table 3: Consonants

UNI	TRA	Dvg/San	Ben	Gmu/Pun	Ory	Sin[d]	Kan[c]	Tel[c]	Mlm	Tam	Tib[d]
KA	k	क क	ক	ક ક	କ	ක	ಕ	క	ക	க	ཀ
KHA	kh	ख ख	খ	ખ ખ	ଖ	ඛ	ಖ	ఖ	ഖ		ཁ
GA	g	ग ग	গ	ગ ગ	ଗ	ග	ಗ	గ	ഗ		ག
GHA	gh	घ ɦ	ঘ	ઘ ઘ	ଘ	ඝ	ಘ	ఘ	ഘ		གྷ
NGA	ṅ	ङ ङ	ঙ	ઙ ઙ	ଙ	ඞ	ಙ	ఙ	ങ	ங	ང
CA	c	च च	চ	ચ ચ	ଚ	ච	ಚ	చ	ച	ச	ཅ
CHA	ch	छ छ	ছ	છ છ	ଛ	ඡ	ಛ	ఛ	ഛ		
JA	j	ज ज	জ	જ જ	ଜ	ජ	ಜ	జ	ജ	ஜ	ཇ
JHA	jh	झ झ	ঝ	ઝ ઝ	ଝ	ඣ	ಝ	ఝ	ഝ		
NYA	ñ	ञ ञ	ঞ	ઞ ઞ	ଞ	ඤ	ಞ	ఞ	ഞ	ஞ	ཉ
TTA	ṭ	ट ट	ট	ટ ટ	ଟ	ට	ಟ	ట	ട	ட	ཊ
TTHA	ṭh	ठ ठ	ঠ	ઠ ઠ	ଠ	ඨ	ಠ	ఠ	ഠ		ཋ
DDA	ḍ	ड ड	ড	ડ ડ	ଡ	ඩ	ಡ	డ	ഡ		ཌ
DDHA	ḍh	ढ ढ	ঢ	ઢ ઢ	ଢ	ඪ	ಢ	ఢ	ഢ		ཌྷ
NNA	ṇ	ण ण	ণ	ણ ણ	ଣ	ණ	ಣ	ణ	ണ	ண	ཎ
TA	t	त त	ত	ત ત	ତ	ත	ತ	త	ത	த	ཏ
THA	th	थ थ	থ	થ થ	ଥ	ථ	ಥ	థ	ഥ		ཐ
DA	d	द द	দ	દ દ	ଦ	ද	ದ	ద	ദ		ད
DHA	dh	ध ɦ	ধ	ધ ધ	ଧ	ධ	ಧ	ధ	ധ		དྷ
NA	n	न न	ন	ન ન	ନ	න	ನ	న	ന	ந	ན
NNNA	n̲	ऩ								ண	
PA	p	प ɒ	প	પ પ	ପ	ප	ಪ	ప	പ	ப	པ
PHA	ph	फ फ	ফ	ફ ફ	ଫ	ඵ	ಫ	ఫ	ഫ		ཕ
BA	b	ब ब	ব	બ બ	ବ	බ	ಬ	బ	ബ		བ
BHA	bh	भ भ	ভ	ભ ભ	ଭ	භ	ಭ	భ	ഭ		བྷ
MA	m	म म	ম	મ મ	ମ	ම	ಮ	మ	മ	ம	མ
YA	y	य ɒ	য	ય ય	ଯ	ය	ಯ	య	യ	ய	ཡ
RA	r	र ɾ	র	ર ર	ର	ර	ರ	ర	ര	ர	ར
RRA	r̲	ऱ					ೱ	ఱ	ഩ	ற	
LA	l	ल ल	ল	લ લ	ଲ	ල	ಲ	ల	ല	ல	ལ
LLA	l̲	ळ ऺ		ૹ	ଳ	ළ	ೞ	ళ	ള	ள	
LLLA	l̲̲	ऴ							ഴ	ழ	
VA	v	व ं		વ વ	ୱ	ව	ವ	వ	വ	வ	ཝ
SHA	ś	श श	শ	શ શ	ଶ	ශ	ಶ	శ	ശ		ཤ
SSA	ṣ	ष ः	ষ		ଷ	ෂ	ಷ	ష	ഷ	ஷ	ཥ
SA	s	स ऱ	স	સ સ	ସ	ස	ಸ	స	സ	ஸ	ས
HA	h	ह ह	হ	હ હ	ହ	හ	ಹ	హ	ഹ	ஹ	ཧ

UNI – Unicode character names.

TRA – for Romanized transliteration (produced by Dominik Wujastyk)
the Type 1 font CS Bitstream Charter is used.

[c] The composite glyphs are not completed here.

[d] Sinhalese and Tibetan character sets are significantly different from the Indic repertoire.

TEX Live under Windows: what's new with the 7th edition?

Fabrice Popineau
SUPELEC
2, rue E. Belin
57070 Metz
FRANCE
fabrice.popineau@supelec.fr

Abstract

The 7th edition of TEX Live under Windows has some new features that are explained in this paper. Especially notable are two experiments to extend Kpathsea beyond its original abilities:

- sharing Kpathsea data-structures between several processes for faster processing,
- allow url's in client programs as well as filenames whenever they ask Kpathsea to open a file.

Additional points about the `TeXSetup.exe` program and the evolution of other parts of the distribution will also be discussed.

Extensions to Kpathsea

Rationale Kpathsea is at the heart of any Web2C based TEX distribution. The idea of gathering all the services needed by the TEX family programs around a common API was a great step forward, compared to the previous TEX distributions. Kpathsea features have been quite stable since version 3: there have been only minor changes. The syntax of the `texmf.cnf` file is the same, the file searching algorithm has converged to the current one, which may not be entirely satisfactory, but which is at least stable.

One of the Web2C features – the `\write18` command – triggered new uses of the whole TEX system. Specifically, it became possible to make `tex` call `metapost`, which will on its turn call `tex` again to typeset labels. As one document can hold many hundreds of metapost figures, we may wonder about the overhead when starting Kpathsea. In fact, we already noticed that the Kpathsea initialization time is far from insignificant. If we take the full TEX Live installation as a reference, then we have quite a long `texmf.cnf` file to parse and a huge `ls-R` hash-table to build, which are responsible for most of this initialization time. So for a given job, which needs to run `tex` and `metapost` in sequence for hundreds of figures, we can wonder if we could cut down the processing time by avoiding to repeat this initialization sequence: most of the data structures that Kpathsea will build across a single job will be the same for each instance of the programs.

A second extant issue with Kpathsea is how far the path notion extends. From the beginning, Kpathsea was written to handle any kind of paths for any kind of operating systems: IBM MVS, DEC VMS, Unix, Amiga, DOS, OS/2, Win32. Not all of them are still actively supported, but they could be so in principle. Among them, Unix has probably the simplest, most regular path syntax.

But supporting Win32 means that UNC share names like `//Server/SharedDirectory/` have to be supported, as well as `c:/Program Files/TeXLive/texmf/tex/latex/base/article.cls`. Looking at these pathname examples, we can see that each of them can be divided into 3 parts: a source (server, shared name, drive), a path relative to the source and the file name. So given that the URL syntax also follows the same pattern, we can wonder why it could not be supported by Kpathsea.

A bare implementation of both features is active in the Win32 version of Kpathsea which is available on TEX Live 7. The description of their implementation and related problems follows. It is important to keep in mind that all the changes with report to the standard TEX Live sources have been made with simplicity in mind. Sometimes better solutions could have been found but at the price of more extensive changes or code rewriting. All of the TEX Live specific Win32 changes are provided by the `/source/source-win32-patch.tar.bz2` file on the cdrom.

Extending the "path" notion What do we want to achieve in extending Kpathsea towards the Internet? The basic usage could be something like this:

```
\includegraphics{http://server.net/image.jpg}
```

It could also be interesting for files included in a document to be taken from the Internet. We can wonder how long such a url might last (the Internet is moving so fast), but in the meantime more and more people are processing XML documents using TEX and so it can be useful to retrieve XML fragments off the Internet. Given this, we need to cope with only 3 kinds of url's: `file://`, `ftp://`, `http://`. The others are not useful. The first one is not a problem because it is only syntactic sugar for local files.

So we want at least that any file opened for reading by TEX can be replaced by an `ftp://` or `http://` url. What does this imply from the file point of view? Playing with remote files introduces unknown behaviour because the files may not be available. Moreover, both ftp and http protocols have different features. The http protocol allows for retrieving file information (availability, size) without transferring the actual file, whereas the ftp protocol cannot feasibly do this.[1]

The simplest solution for implementing remote file access is to use a third party library that downloads the file locally. Downloading the file at opening time will ensure future availability. All the remote files are given temporary names and the association between url's and temporary names is stored in a hash-table. When Kpathsea exits, all such temporary files are removed. This is far from optimal, but safe enough for testing the feature. We could enhance the process by managing a cache of downloaded files, avoiding download of files that are already present unless they are too old and so on, but this is probably not a job for Kpathsea: if such a feature has to be used intensively, then it would be better to have a high-performance cache system for your whole Internet connection.

The question is now the following one: where to hook in the function that will retrieve remote files? Given that we essentially expect TEX and friends to be the programs using remote files, we could hook into the `web2c/lib/openclose.c` file which holds the input/output functions for TEX programs. However there are also reasons to hook inside Kpathsea:

- we need to extend path parsing and this is Kpathsea job;
- we can wonder about the opportunity of making Kpathsea Internet aware and be able to set up a remote texmf tree for example;
- other programs than TEX engines could benefit from this feature;
- under Win32, it is easier to replace `kpathsea.dll` if we want to enhance it or fix it.

So we chose to make the changes inside Kpathsea rather than inside the TEX engines. The internals of Kpathsea have already a few macros and functions used to parse various kinds of paths, so adding a couple of macros and a few cases inside these functions was easy. Given the fact we want to download remote files as soon as they are accessed, we trap the `fopen()` call and hook file downloading there. This way, any Kpathsea file can be replaced by a url. This is enough to provide absolute remote file retrieval. However it is not enough yet to provide remote file searching. This would require to trap also the `opendir()` and `readdir()` calls. Although internet aware versions of these calls are provided by the Win32 WinInet API, this change has not been implemented yet[2].

The only remaining problem was to find a reliable library to download files. There are several choices:

libwww the W3 Internet protocol library, which is Unix and Win32 compatible. After quick testing under Win32, file downloading was unreliable and blocked sometimes. Moreover, even after careful documentation reading, we have never found out how to display download progress, although it is said to be possible;

libcurl this library is provided together with a command line tool and seemed to be very handy. It has been confirmed at least under Win32 where it worked like a charm at first try;

wininet the Microsoft Internet library provided with Internet Explorer 5 and later. This is solid and has a high-level API, but Win32 only;

sockets raw protocol handling is possible too but if we want reliability, better to rely on some higher level library.

Ideally, it should be possible to plug any new library into the system and choose the one used from `texmf.cnf`. We need only one function to ensure the compatibility layer:

```
int get_url_to_file (
  /* the url of the file to download */
```

[1] Retrieving the file size requires parsing a directory listing, but there is no standard for this listing.

[2] In fact, because of the overhead introduced by initializing `ls-R` hash-tables for a remote texmf tree, this could be interesting in the larger framework of a Kpathsea server. See Karel Skoupy's talk on this topic in this same proceedings.

```
char *_url,
/* the temporary filename */
char *_filename,
/* if we know the file size */
int expected_size,
/* the function to log the download */
pfnLog log,
/* the function
   to report download progress */
pfnProgress progress,
/* the download method to use */
int method
                  );
```

According to the method selected one or another library will be used to download the file. Currently, wininet is hardwired as the selected method but work is under progress to provide the same feature with libcurl under Unix and as an alternative to wininet under Win32.

The feature works quite well and is easy to use. Enhancing it will require a careful analysis of its uses. For example, it is common usage to open a file to test its availability. In this context, the file will be downloaded unconditionally, which we might want to avoid.

Kpathsea and reentrance The initialization time problem would have been easily solved if Kpathsea had been reentrant, but this is far from being the case. Kpathsea has been designed for quite a few years now and that feature was certainly not given a high priority if envisaged at all. Making Kpathsea fully reentrant is certainly not an easy task without major rewriting: that would mean for example that different programs – say `latex` and `context` – could call it simultaneously to look for files and it would be able to answer both of them. For the moment, unfortunately, the calling program, upon which search paths depend, is defined at initialization time, and the current data structures make it difficult to redefine it afterwards.[3]

Another point with initialization time is that it is even longer under Win32 because the file systems (FAT32, NTFS) are case-insensitive (though they preserve case). Hence, filenames read from the `ls-R` files are stored in the hash-table after conversion to uppercase. But since Win32 is able to handle Unicode and MBCS, the conversion realized by the function `toupper()` is as slow as you might expect (it has been measured several times slower than the same function provided by the GNU C library which did not handle MBCS or Unicode).

[3] Actually, it can be done and the C version of the `fmtutil.exe` program does it, but it can work only in limited cases.

Solving our startup time problem is, however, quite simple. All we need is to prevent Kpathsea from rebuilding all of its hash-tables across a single run of several Kpathsea-linked programs. Here is the list of these hash-tables:

1. configuration parameters, essentially every variable/value pair read from the `texmf.cnf` file;
2. `ls-R` database, the huge (around 40000 for TEX Live) list of filenames and paths;
3. alias database, this is a list of alias names for a few files;
4. fontmap database;
5. links database, this is internal data used to accelerate the recursive search on paths;
6. symbols database, this is the list of pairs of variables and values used to emulate the various `mktex...` shell scripts in C (Win32 specific);
7. remote files database, the remote files that have been downloaded from the Internet and stored locally under a temporary name (see previous section).

Storing all these hash-tables into a shared memory block allows us to avoid rebuilding them. If the block does not exist yet, it is then initialized. If it is there, then we can bypass Kpathsea initialization and use the already built hash-tables. Managing a shared memory block has some drawbacks however:

- the shared memory block is allocated once and for all, all you can do after it is built is release it; it is not extensible. This is because it needs to be allocated at the maximum size that will be needed, or else it is quite inefficient to resume from the situation (see below);
- all pointers must reference data relative to the beginning of the shared memory block: there is no way to know the address of this block for each of the processes using it;
- none of the standard malloc facilities (`malloc`, `free`, `realloc`) are provided to play with memory inside this shared memory block. Fortunately, managing the hash-tables only requires that we allocate memory. Although occasionally we might need to remove an item from a hash-table, we can safely ignore freeing this memory because the situation does not occur frequently.

Managing data inside the memory block required a few changes to Kpathsea functions. All data are reallocated inside the shared memory block, so there is no point in storing a pointer to preallocated memory as was the case in a few places. When using the hash-tables, all values returned are

`const char *`: it is the user's responsibility to duplicate them to become writable. All of this has been done by introducing a new module replacing the `kpathsea/hash.c` one, with a slightly different, more regular API for the hash-tables functions. Then, all the calling functions were adapted to this new API. Managing relative pointers inside the shared memory block has been done by macros, but all pointers returned by the hash-tables function are absolute addresses inside the virtual process space.

Shared memory blocks are not easily extendable. Given that our block is identified by its name, extending it requires allocating a second one with a new name, copying data from the old one to the second one, releasing the old one, creating a third one with the old name but larger than the first one, and finally copying the data from the second one to the third for the second time! Moreover, allocating a new block supposes we can propagate the new base address which can become tricky. In fact, we are only able to extend the shared memory block at initialization time (so no propagation needed) using the previous mechanism. Shared memory block extension will occur only if the block is filled when reading the hash-tables, or if there is not enough free space after initialization is done.

The `mktexupd` program has been enhanced to add directly the new file inside the already loaded hash-tables because it is merely an addition to the existing list of files. But to cleanly allow to rebuild all data structures for `mktexlsr` while the shared block is loaded would require careful redesign of Kpathsea (especially to be able to reinitialize it at any point).

One side effect of using this feature has both a positive and a negative aspect. If you have some program like the `windvi` viewer which is using Kpathsea, then the Kpathsea dll will not be unloaded until the viewer is closed. As long as an instance of the dll is running, the shared memory block will remain. So the shared memory block will be available for all TEX (and other Kpathsea-linked programs) runs that will occur and you will gain initialization time at every run. This is the positive aspect. The negative aspect is that you need to close all programs linked to Kpathsea whenever you want to change your configuration in the `texmf.cnf` file or whenever you need to run `mktexlsr`. Again, the hash-tables are built once and for all, and given the lack of real memory allocation operations inside this shared block, we cannot rebuild them easily unless *we shut down all the programs using Kpathsea.*[4]

The fact that `ls-R` files are not read again when the Kpathsea dll is loaded is minor inconvenience. It could be part of the managing environment to ensure that all processes are shutdown when `mktexlsr` is run. The `kpsecheck` tool is provided and reports about the shared memory block usage (see the section Other programs below).

Windows-specific TEX Live features

Small enhancements to Web2C MiKTEX has had for some time options to the TEX family of programs that have not been available under Web2C. In order to enhance compatibility between both distributions, we added these options to the Win32 Web2C programs too.

The most noticeable one is the `-job-name` option. Using the standard Web2C TEX program (version 7.3.7), there is some inconsistency with the `-fmt` option:

- in virtex (or normal) mode, this option tells TEX to load the format specified,
- in initex mode, it requires TEX to dump the format under the specified name.

But we could also require to load some format and specify the dump name. In fact, the `-job-name` option solves this problem, because it allows to change the internal `\jobname` value, which is used both in ini and vir modes. So under Win32, the `-fmt` option has only one meaning: preload the specified format and the `-job-name` option can be used to change the base name of the files output by TEX. All of this is whether you are in ini mode or not.

The other options introduced under Win32 are:

-halt-on-error stop at the first error;

-job-time=FILENAME set the job time by taking FILENAME's timestamp as the reference;

-output-directory=DIR use DIR as the directory to write files to;

-time-statistics print processing time statistics about the current job.

The `TeXSetup.exe` program `TeXSetup.exe` has been made more reliable, especially for installation under Win2K/XP. Installation can be done for all users, but then you need to be a Power User or an Administrator, or install it for single users. The default location is `c:\Program Files\TeXLive` to

[4] Apart from `windvi`, there is another resident program which is linked to Kpathsea: it is `ispell`, the spell checker. If you run it from Emacs, you are bound to forget that `ispell` is running in the background.

be Windows compliant, but installing there can be done only by Power Users or better. If you have only standard rights, you can't install under the `c:\Program Files` default location but you can install in `c:\TeXLive` for example.

Win2K/XP rights behaviour may not be very intuitive, especially if you come from the Unix world. Any object rights are inherited from its container object at creation time, except if specified otherwise. The problem with TEX and especially when `mktexpk` build font files, is that when an object is moved, this object keeps its original rights. So if we apply this rule to `mktexpk`, the `.pk` is built by user A in the user temporary directory, so inherits user A's rights, then it is moved in the `$VARTEXMF` directory, hoping it will be available for everybody. But the file has kept user A's rights, so another user has no rights to access it. Worse, another user can't even overwrite it: the file is there and is neither accessible neither removable. To overcome this problem, we came up with this solution:

- at setup time, `TeXSetup.exe` creates the temporary directory `c:\Program Files\TeXLive\temp` directory and assigns it full access rights for everybody;
- the `TEXMFTEMP` variable is assigned the value of the temporary directory;
- at initialization time, Kpathsea substitutes the value of `TEXMFTEMP` for the standard `TEMP` value.

This way, all temporary files are created in the `TEXMFTEMP` directory, so they are given full access rights to everybody. Font files will keep these full access rights when moved to the `VARTEXMF` directory. That means everybody can remove them too, but this is not a problem since these files will be rebuilt at runtime when they are not available.

TEX Live 7 introduces so-called schemes of installation. Previously, we had collections and packages. In order to make things easier, we can now require that some predefined set of collections and packages be installed. Such a set is called a scheme. We have defined a couple of them: texlive-basic, texlive-recommended, texlive-full for standard texlive distributions of different sizes, but there are also the GUST scheme, the GUTenberg scheme and the XML scheme. What is interesting is that you can run batch installation using this kind of command:

```
c:\>TeXSetup --scheme=texlive-full --quick
```

It is easy to add new schemes by creating a new TPM file in the `texmf/tpm/schemes` directory. Complete documentation for the `TeXSetup.exe` program is provided with the TEX Live manual.

Other programs TEX Live holds quite many Perl scripts which are widely used: `texexec`, `texutil`, the new `updmap` (rewritten in Perl for Win32 from the original shell script version), etc. Up to now, you had to have Perl installled to be able to use them. Unfortunately, this is not an ideal solution, since:

- Perl is not a standard part of Windows;
- Perl is not available on the TEX Live CD-ROM because of disk space, although it is available for installation from the Internet as part of the Win32 support packages;
- if Perl is not installed, the scripts are not usable off the CD-ROM.

So with TEX Live 7, we decided to compile all the Perl scripts into `.exe` files using the ActiveState Perl compiler. There are drawbacks in doing this:

- the size of `.exe` files is quite large, even if the `perl56.dll` needed to run them is provided in a common `bin/win32` TEX Live directory and not included in the `.exe` compiled file,
- users cannot easily upgrade the Perl scripts, because they would need to be recompiled.

However, this is currently the only way to run those scripts off the CD-ROM on a machine where Perl is not installed.

Another new point with TEX Live 7 is the ability to build format files at run time. This was quite easy to setup inside Kpathsea because the mechanism is the same as the one used for fonts: when the requested file is missing, the specified command is run to try to build it. So in this case, we only needed to add the new `mktexfmt` command which actually behaves like the `fmtutil` one. The only difference is in the calling sequence:

```
c:\>fmtutil --byfmt=latex
```

versus

```
c:\>mktexfmt latex
```

and also the |mktexfmt| needs to write the path of the generated file to `stdout`. This `mktexfmt` command was easy to build out of the `fmtutil` shell script (Unix) and the `fmtutil.exe` C program (Win32). There are several benefits in doing this:

- no need to run `fmtutil -all` at the end of installation anymore (even if it is still done), since format files will be built on demand;
- fewer `.bat` files for all kinds of formats (those `.bat` files used to build the format file when it was not found),; for example, you only need to copy `tex.exe` to `mllatex.exe` and the command `mktexfmt mllatex` will be run if the `mllatex.fmt` format file is not found, provided your `fmtutil.cnf` file mentions it. Then, the

execution will continue. It even works for nested dependencies between format files.

Lastly, there is a new tool call `kpsecheck` which can report on a few aspects of TEX Live setup:

- the state of the Kpathsea shared memory block (in a future version this should include the processes using it);
- the Ghostscript location: Ghostscript is used in many places and it is handy to make those programs find it automatically;
- duplicates among the files stored in the hash-tables. Called with no argument it will report all the duplicates, but many files have the same name (`README`, etc.). So you have the option to include or exclude files based on globbing:

```
c:\>kpsecheck -multiple-occurences \
              -include=*.sty
```

to find all duplicates among `.sty` files.

What is coming for the 8th edition?

It took a few years to stabilize the whole Win32 TEX Live. A few points are still missing, the most obvious involving the TPM files and the `windvi` previewer.

Talking about the previewer, the main missing features are:

- source specials support,
- Type1 and TTF font files support,
- safe printing and `dvips` printing.

Source specials support and Type1 font support are available from XDvi now. So importing this into Windvi will be quite easy.[5] TTF font files support will require a bit more work but it should follow the same global scheme as for Type1 fonts. Printing is an area that will require careful testing. The most annoying point is the huge spool files potentially created. This has been partially avoided by doing banding, at the cost of slowness.

[5] A first version of `windvi` supporting these features will be demonstrated at the conference.

The setup scheme could be enhanced in various ways. The granularity of TPM files is not optimal: binaries should be split into smaller packages. There are annoying dependencies between some packages (for instance, `aeguill` relies on files provided by other unrelated packages, like the Polish fonts). It is difficult to automate the construction of these packages from what is available on CTAN, even if Sebastian Rahtz has done a great work in this area. Probably we need a whole new infrastucture to submit packages to CTAN but that is another story.

Finally, starting next October, we will work on a project of tight integration between XEmacs and TEX in order to provide an easy-to-use out-of-the-box word processing tool. The goal of this project funded by the French Ministry for Education is to draw certain kind of people (mainly mathematics and physics teachers) towards TEX, on the premise that what refrains people to use TEX is not the TEX input syntax, but the complexity of installing and maintaining a TEX distribution. So we will do our best to provide a well documented, regular TEX system with integrated viewer. More to come about this project next year!

References

Popineau, F. "fpTEX: a win32 port of teTEX". In *TUGboat*, volume 20. TEX Users Group, 1999. Proceedings of the 20th Annual Meeting of the TEX Users Group, Vancouver.

Popineau, F. "Directions for the TEXLive Software". In *Proceedings of the EuroTEX 2001 Conference, Kerkrade, the Netherlands*, pages 151 – 161. 2001.

Rahtz, Sebastian. "The TEX Live Guide, 7th edition". Available on the TEX Live 7 CD-ROM, 2002.

Skoupy, Karel. "TEX file server". In *Proceedings of the 23rd Annual Meeting and Conference of the TEX Users Group*. 2002.

Catching up to Unicode*

Roozbeh Pournader
Computing Center
Sharif University of Technology
Azadi Avenue
Tehran, Iran
roozbeh@sharif.edu
http://sina.sharif.edu/~roozbeh/

Abstract

Unicode, the universal character set standard first published in 1991, has changed dramatically in its more than ten years of development, trying to achieve maximum interoperability of internationalized text between different platforms. In the meanwhile, TEX, and its companion production tools, have stuck loyally to their roots of special formats and traditions known only to the TEX community, rather ignoring this moving target.

This paper will try to draw a general outline of the Unicode standard in its present situation, emphasizing on its recently introduced features. It will also try to specify new requirements for Omega, in order to make it usable in the developing realm of *standard renderings*.

Introduction

The Unicode Standard (The Unicode Consortium, 2000, as amended by Unicode Editorial Committee, 2001, and Unicode Editorial Committee, 2002), and the big family gathered around it, ranging from MathML (Carlisle et al., 2001) to OpenType (Adobe Systems Incorporated and Microsoft Corporation, 2001), have been successful in making document interchange truly global. Now, you don't need to worry about the very basics of how to encode your text if you want to develop an application handling Lao, or send an email containing APL symbols; you won't need to educate the user base about semantic markup, as even Microsoft is now recommending XML; and you don't even need to develop new tools for your basic Syriac text processing needs, IBM has already developed many general purpose ones if you can't find some suitable ones in your Linux machine, which you only need to configure.

The Unicode standard, while keeping the same encoding model of the beginning days, has become very complicated recently, because of the advanced requirements of text processing applications and the natural languages and scripts themselves. Also, because of the wide deployment of Unicode by Microsoft, a Unicode contributor, starting with Windows 2000 and Office 2000 series of products, many loopholes and implementation difficulties have been discovered, and fixed in the standard.

Unfortunately, Microsoft has got some of the ideas wrong, but fortunately, many of us don't live in a Microsoft world. Linux and GNU communities, with their affinity for standards, and now backed by corporations like IBM and Sun, have helped implement Unicode in a compliant way. With an out of the box installation of Red Hat Linux 7.3, you can now edit a Unicode text document in Ethiopic containing some Hebrew, some Braille, and a few chess symbols using vim under xterm[2]. But sadly, very few tools exist to help you make a typographically beautiful printout of the file.

The current Unicode

Putting history aside, Unicode, in its latest 3.2 version, is a character set assigning a number from 0 to $\texttt{10FFFF}_{16} = 17 \times 2^{16} - 1 = 1,114,111$ to each character[3]. (We will be using the notation U+20A8 to refer to the character coded as $\texttt{20A8}_{16}$, which (as it happens) is a RUPEE SIGN). The characters are distributed in blocks of related functionality, such as

* Preparation of this paper, its prerequisite research, and its presentation at TUG 2002 have been supported by the FarsiWeb Project, Science and Arts Foundation (http://www.farsiweb.info/).

[2] The same is of course possible under Microsoft Windows XP if you have the appropriate fonts and keyboard mapping programs.

[3] At present, the standard intends never to change this upper limit, barring extraterrestial scripts!

a script like Cyrillic or a common usage like Mathematical Operators.

Unicode does (and will) not encode any presentation forms (also known as *glyphs*) unless for backward compatibility with legacy character sets. So, you will not see four different characters for the Syriac letter Beth, but only one.

On the other hand, there are many kinds of special characters encoded, from control characters for specifying the shape of newly-invented CJK ideographs to weather forecast symbols. There are also more than 70,000 characters reserved for private use, which lets two parties interchange text without requiring the encoding of their limited-use characters by the standard.

Unicode also assigns many normative and informative properties to each character, together with descriptions of characters and blocks. For example, the Arabic block contains a very exact description of a minimum Arabic contextual shaping algorithm, and the General Punctuations block contains explanatory text for each punctuation mark to make sure that not a single character is misunderstood. A list of some character properties together with their description can be found in Table 1.

The Unicode Consortium is quite proactive, as well as responsive to requests, in encoding new characters and redefining character properties. Its Unicode Technical Committee meets quarterly, and tries its best in both keeping stability and backward compatibility, and fixing any existing mistakes. The author has had a pleasing experience with his proposals to the committee.

Character Properties Some normative properties worth special note are *canonical decompositions*, *compatibility decompositions*, and *combining classes*. For instance, Unicode has two ways to encode 'ä', one being the character U+00E4, Unicode Small Letter A With Diaeresis, and the other the sequence of characters ⟨U+0061, U+0308⟩, which is actually a Latin Small Letter A, followed by a Combining Diaeresis. To help applications, Unicode specifies a mechanism for decomposing the former character to the latter sequence, to check equivalence. The situation will get more complicated when you consider the case of double or multiple accents: a Combining Cedilla can be equally followed or preceded by a Combining Tilde, but you can't say that about a Combining Tilde and a Combining Acute Accent, where an interchange will result in different renderings. So, there will be a need for combining classes, which are numbers assigned to each combining character (equal numbers specify non-interchangeable order). The other properties, compatibility decompositions, are decompositions specifying approximate equivalence, like the character U+210E, Planck Constant (h) which is specified to be compatibly equivalent to U+0068, Latin Small Letter H, with only a font difference.

Based on these two ideas, come Unicode *normalization forms* (see Davis and Dürst, 2002). The normalization forms are there to help one do binary checking instead of heavy table lookups. Once the text is in a normalization form such as NFD, in which all precomposed characters are decomposed based on their canonical decompositions, and combining characters sorted based on their combining classes[4], equivalent strings will become equal and it will be good old days again, where you could check string equivalence using the C function `strcmp` or search your files with the UNIX tool `grep`.

In practice, it is Normalization Form C (NFC), a form that re-composes the characters after decomposition, which is the most important form. Having maximum compatibility with legacy character sets, and requiring a simpler rendering logic which eases implementation in portable devices, NFC is referenced frequently in the Character Model for World Wide Web (Dürst et al., 2002) as the normalization form required for all web content. It is also the preferred way for encoding text files and file names in UNIX (except in Mac OS X, where NFD is used). Another form, NFKC, which additionally uses compatibility decompositions, is a requirement of IETF's International Domain Names in Applications (Fältström et al., 2002).

Another important character property, which is absolutely required in the area the author lives, is the *bidirectional category*. These are categories assigned to each character specifying its behavior in mixed right-to-left and left-to-right text, which is common in scripts like Arabic and Hebrew. Characters are divided to classes like *left-to-right*, *right-to-left*, *European number*, *Arabic number*, *common separator*, *white space*, ... which are used in a carefully specified Bidirectional Algorithm (Davis, 2002) to reorder a *logically-ordered* stream of characters to a *visually-ordered* one.

There are also many other character properties, but worth special notice is that breaking almost any of them will make your application non-compliant. You cannot have your Arabic Comma behave as a strictly *right-to-left* character, render two canonically equivalent strings in two different ways, or

[4] The compatibility decompositions will be ignored.

Code Point	The numeric code assigned to the character: '2057' for the Quadruple Prime. This can never be changed.
Character Name	A name only using uppercase Latin letters, digits, hyphen and space. This can never be changed.
General Category	A category describing the general behavior of the character: 'Sc' (Symbol, Currency) for the Euro Sign.
Canonical Combining Classes	A class used for ordering combining marks and accents after a base letter: '230' (Above) for Combining Tilde and Combining Grave Accent and '202' (Below Attached) for Combining Cedilla.
Bidirectional Category	Categories specifying the behavior of the character in a bidirectional context: 'AL' (Right-to-Left Arabic) for Arabic Letter Beh.
Character Decomposition Mapping	A decomposition of the character to a sequence of others: '0075 0302' (⟨Latin Small Letter U, Combining Circumflex Accent⟩) for Latin Small Letter U With Circumflex, and '⟨super⟩ 0054 004D' for Trade Mark Sign (^{TM}). There are many stability requirements for the character decompositions (Davis and Dürst, 2002).
Decimal Digit Value	A numeric value for digits: '3' for Arabic-Indic Digit Three.
Numeric Value	A numeric value for characters that specify numbers: '1/5' for Vulgar Fraction One Fifth.
Mirrored	Specifies if the character image should be mirrored in text laid out from right to left: 'Y' (Yes) for Element Of.
Uppercase Mapping	A one-to-one mapping for converting letters to uppercase: '053F' (Armenian Capital Letter Ken) for Armenian Small Letter Ken (for full case mappings, see Davis, 2001).
Lowercase Mapping	Similar to uppercase mapping, providing lowercase forms: '1043E' (Deseret Small Letter Jee) for Deseret Capital Letter Jee.
Titlecase Mapping	Similar to uppercase mapping, providing titlecase forms: '01C8' (Latin Capital Letter L With Small Letter J for Latin Small Letter Lj).
Arabic Joining Type	Specifies the shaping behavior of an Arabic or Syriac letter: 'D' (dual-joining) for Arabic Letter Sheen.
Arabic Joining Group	Specifies a group for each Arabic and Syriac letter, specifying the letter it will be shaped like: 'SEEN' for Arabic Letter Sheen.
Line Breaking Property	Specifies how the character behaves relative to line breaking: 'BA' (Break Opportunity After) for En Dash (see Freytag, 2002 for more details).
Special Casing Properties	Uppercase, lowercase, and titlecase mapping for languages that handle the case differently: '0130' (Latin Capital Letter I With Dot Above) for Latin Small Letter I in Turkish and Azeri contexts.

Table 1: Some of the various standard properties of Unicode characters.

use some unassigned code points for document interchange, and claim compliance at the same time. The user undoubtedly doesn't like her text to have different meanings in different applications.

Interesting Features These features are some of the interesting ones for typographically and semantically oriented mind sets. Some appear only in later versions of the standard.

- Having different characters for letters that look the same in uppercase but have different lowercase forms. Also, providing different characters for mathematically different forms of letters like the Greek small Phi.
- The ability to recommend for or against automatic ligation in special circumstances, using the characters Zero Width Joiner and Zero Width Non-Joiner.
- Specifying *line breaking* properties for characters, to help recommending for or against a line break.
- Specifying a *mirroring* behavior for some characters like the opening parenthesis, whose image should be mirrored in a right-to-left context.
- The ability to encode implicit mathematical semantics, like an invisible times operator or an invisible comma separator, and even a smart Division Slash for in-line fractions like $22/7$.
- Having characters for almost every symbol required in mathematics typesetting from four combining characters for different forms of `\not` to a character for rare symbols like Mathematical Sans-Serif Bold Small Xi, and even bracket, brace, and parenthesis fragments, like those available in TEX's cmex font.

OpenType

Because Unicode does not provide standard codes for glyphs, but rather is interested only in characters, there is a need for a standard to fill the gap, that is, provide a mechanism to map characters to glyphs in a font. There are three available: Microsoft and Adobe OpenType, Apple AAT, and SIL Graphite (Correll, 2000). Of these, OpenType, a superset of TrueType, has become much more successful, with many outside implementations including some from IBM and the FreeType and GNOME projects. It is becoming the *de facto* standard not because the specification is very clear or technically supreme, but since many high quality fonts exist in the format[5]. Actually, when you get to scripts a little more complex than European ones, say Devanagari or Khmer, OpenType is the only font format in which you will be able to find a couple of usable fonts.

Unlike the AAT and Graphite formats mentioned above, OpenType fonts do not encode the basic visual behavior of the characters in the scripts they support. It's the text layout engine that should know about the script, and the font will only provide the minimum needed information for locating various presentation forms, kerning, positioning accents and marks, and ligating (among others). For example, a font will not include any information about the contextual shaping behavior of U+0649, Arabic Letter Alef Maksura, like if it is a right-joining letter or a dual-joining one[6]. It will only specify that a final presentation form of the letter can be found at a certain glyph position.

OpenType introduces new font tables once in a while (even allowing font developers to register some with the specification owners), specifying features like Hebrew mark positioning or Arabic swash forms. This means that the font format is both backward compatible (old engines won't know about advanced features, but can still render the text using the older features available in the font), and extensible (an overlooked feature in a minority script can be requested by anyone and included in the next version of the specification, and everyone will be free to implement or ignore it).

Other Friends

Unicode and OpenType are joined by standards like Adobe PDF (Adobe Systems Incorporated, 2001) which in the latest version provides some mechanisms for having a glyph stream and a Unicode character stream specify a document's visual layout and semantic content in collaboration, W3C Cascading Style Sheets (Bos et al., 1998; Suignard and Lilley, 2001) that includes mechanisms to guide automatic selection of fonts for Unicode characters from various scripts that appear in a single text document, and MathML, with almost all of its symbols now in Unicode, which will happily let you cut and paste mathematical formulas between different applications.

[5] For example, Adobe recently re-released all of their professional fonts in the OpenType format, abandoning both PostScript Type 1 and Multiple Master for new fonts.

[6] Actually, the joining class of this letter was changed between Unicode versions 3.0 and 3.0.1.

Many other standards have also emerged, both using Unicode's power in supporting technical symbols and minority scripts, and learning from its successful unifying experience. Fortunately, these standards are not only on paper, but they are followed and being implemented carefully by both major software houses and Open Source and Free Software communities. The author is specially interested in the linux-utf8 mailing list, where discussions about implementing Unicode-related technologies in GNU, Linux and UNIX platforms happen (subscription information is available at `http://www.cl.cam.ac.uk/~mgk25/unicode.html#lists`).

Where We Stand

In this picture, TEX and friends are left far behind the majority of the world. We still mostly use our own font formats, METAFONT, PostScript Type 1, TFM, and PK instead of OpenType (for outline fonts) or pcf and bdf (for bitmap fonts)[7]; the TEX output format DVI is still in very wide use, whereas the rest of the world primarily uses PDF; and almost all TEX friends have their own traditional command sequences with a very loose syntax, in contrast with a world community united on XML. In short, they just don't fit!

This should not be compared to the beginning years, when the TEX community needed to invent formats for technologies that were becoming public for the first time, and had the chance of making them *de facto* standards. Many components can (and should) now be replaced with their widely recognized and recommended equivalents like XSL Formatting Objects (Adler et al., 2001), to have the luxury of high quality typography once again. Contrary to many who insist that they are dead horses not worth more beatings, the author believes that TEX and friends have the ability of reaching many new users if they get modified to support current technologies.

New Requirements for Omega

The author considers Omega as a possible candidate to advance in the field and fill the existing gap. Actually, it is the only candidate among friends like ε-TEX and pdfTEX, which are more typographically oriented. Omega has already passed the hassle of implementing sixteen-bit fonts, pre- and post-processing text filters needed for rendering the so-called complex scripts, and even some MathML. But unfortunately it's not stable, lacks an active development team, and has a rather more academic orientation than desirable for such a project. The worst point of all is that it is being developed in a cathedral model (see Raymond, 2001), which makes it very hard to reach the above goals.

To make Omega usable in the current working environment, the following requirements come to mind:

- Opening the development of Omega, not only accepting contributions from the public, but also making them active in the development process. This will need a few central people at the beginning, with enough time at their hand to sketch and implement a working mechanism for future development.
- Accepting the international standards as they are, and trying to be compliant as much as possible: any problem in standards like Unicode should be taken to the Unicode authors themselves, instead of trying to fix them locally[8]. This will help reach consistency with many other existing tools, or those who may be developed in the future.
- Implementing PDF output (not necessarily a merge with pdfTEX), including Unicode character streams.
- Implementing native OpenType support. Some first milestone for achieving the goal may be providing a tool to convert OpenType tables to ΩTP processes. Also, the current Omega fonts should be converted to the OpenType format[9]. Apart from making Omega able to use almost all of the fonts in the wild, this will be a contribution to the Open Source community who lack good printing engines for texts in non-European scripts.
- Supporting XML and MathML as both input and output formats. Fortunately, some of this has already been implemented.
- Closely tracking new features of Unicode, so as to stay current with the rest of the world. Omega may even be able to act faster than its competitors in this field, especially if it starts to follow the *bazaar* model of development.

[7] Even tools like ttf2pk, which let TEX use TrueType fonts, are not in active development and are based on a frozen branch of the FreeType font engine

[8] The author believes that incorporating a fix which will definitely happen in a yet unpublished version of a standard should be allowed.

[9] Some 'Free UCS Outline Fonts', UCS standing for Universal Character Set which is the alternate name of Unicode, are under development based on existing free outlines including those from Omega. Latest information is available at `http://savannah.gnu.org/projects/freefont`.

Acknowledgments

The author wishes to thank Markus Kuhn and Sebastian Rahtz for initially raising some of the issues discussed in this paper, Martin Dürst, Mark Davis, and many other internationalization evangelists who have helped him understand the development mechanisms of many internationalization-related standards, and participate in their development, and also Karl Berry, for editorial assistance. Final thanks should be given to John Plaice and Yannis Haralambous who created Omega in the first place.

References

Adler, Sharon, A. Berglund, J. Caruso, S. Deach, T. Graham, P. Grosso, E. Gutentag, A. Milowski, S. Parnell, J. Richman, and S. Zilles. "Extensible Stylesheet Language (XSL), Version 1.0". W3C Recommendation, World Wide Web Consortium, 2001. Available from `http://www.w3.org/TR/xsl/`.

Adobe Systems Incorporated. *PDF Reference, Adobe Portable Document Format Version 1.4*. Addison-Wesley, third edition, 2001. Also available in electronic format from `http://partners.adobe.com/asn/developer/acrosdk/docs/filefmtspecs/PDFRe%ference.pdf`.

Adobe Systems Incorporated and Microsoft Corporation. "OpenType Specification Version 1.3". 2001. Available from `http://partners.adobe.com/asn/developer/opentype/main.html` and `http://www.microsoft.com/typography/otspec/default.htm`.

Bos, Bert, H. W. Lie, C. Lilley, and I. Jacobs. "Cascading Style Sheets, level 2". W3C Recommendation, World Wide Web Consortium, 1998. Available from `http://www.w3.org/TR/REC-CSS2`.

Carlisle, David, P. Ion, R. Miner, and N. Poppelier. "Mathematical Markup Language (MathML) Version 2.0". W3C Recommendation, World Wide Web Consortium, 2001. Available from `http://www.w3.org/TR/MathML2`.

Correll, Sharon. "Graphite: An Extensible Rendering Engine for Complex Writing Systems". Technical report, SIL International, 2000. Available from `http://graphite.sil.org/pdf/IUC17_paper.pdf`.

Davis, Mark. "Case Mappings". Unicode Standard Annex #21, The Unicode Consortium, 2001. Available from `http://www.unicode.org/unicode/reports/tr21`.

Davis, Mark. "The Bidirectional Algorithm". Unicode Standard Annex #9, The Unicode Consortium, 2002. Available from `http://www.unicode.org/unicode/reports/tr9/`.

Davis, Mark and M. Dürst. "Unicode Normalization Forms". Unicode Standard Annex #15, The Unicode Consortium, 2002. Available from `http://www.unicode.org/unicode/reports/tr15/`.

Dürst, Martin J., F. Yergeau, R. Ishida, M. Wolf, A. Freytag, and T. Texin. "Character Model for the World Wide Web 1.0". W3C Working Draft, World Wide Web Consortium, 2002. Available from `http://www.w3.org/TR/charmod`.

Fältström, Patrick, P. Hoffman, and A. M. Costello. "Internationalizing Domain Names in Applications (IDNA)". Internet Draft, Internet Engineering Task Force, 2002. Available from `http://www.ietf.org/internet-drafts/draft-ietf-idn-idna-10.txt`.

Freytag, Asmus. "Line Breaking Properties". Unicode Standard Annex #14, The Unicode Consortium, 2002. Available from `http://www.unicode.org/unicode/reports/tr14`.

Raymond, Eric S. *The Cathedral and the Bazaar, Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly, revised edition, 2001. Also available in electronic format at `http://tuxedo.org/~esr/writings/cathedral-bazaar/`.

Suignard, Michel and C. Lilley. "CSS3 module: Fonts". W3C Working Draft, World Wide Web Consortium, 2001. Available from `http://www.w3.org/TR/css3-fonts`.

The Unicode Consortium. *The Unicode Standard, Version 3.0*. Addison-Wesley, 2000. Also available in electronic format from `http://partners.adobe.com/asn/developer/acrosdk/docs/filefmtspecs/PDFReference.pdf`.

Unicode Editorial Committee. "Unicode 3.1". Unicode Standard Annex #27, The Unicode Consortium, 2001. Available from `http://www.unicode.org/unicode/reports/tr27/`.

Unicode Editorial Committee. "Unicode 3.2". Unicode Standard Annex #28, The Unicode Consortium, 2002. Available from `http://www.unicode.org/unicode/reports/tr28/`.

PassiveTEX: an update

Sebastian Rahtz
Oxford University Computing Services
13 Banbury Road
Oxford OX2 6NN
sebastian.rahtz@oucs.ox.ac.uk

Abstract

This paper presents an overview of the development and status of 'PassiveTEX'. PassiveTEX is a library of TEX macros which can be used to process the XML which results from transformation of an XML document to the XSL Formatting Objects vocabulary.

PassiveTEX is a library of TEX macros which can be used to process an XML document which results from a transformation of an XML file to the W3C Formatting Objects (FO) XML vocabulary (see http://www.w3.org/TR/xsl/). The advantage of this is that it provides a rapid development environment for experimenting with XSL FO, using a reliable pre-existing formatter. Since we can now run with the pdfTEX variant of TEX, generating high-quality PDF files in a single operation, PassiveTEX shows how TEX can remain the formatter of choice for XML, while hiding the details of its operation from the user.

How does it work?

PassiveTEX builds on David Carlisle's XML parser written in TEX (XMLTEX), and was developed from my JadeTEX macros for processing DSSSL via Jade. XMLTEX is a highly complex set of TEX macros which parse XML files directly and apply TEX macros to the elements as defined in a configuration file. Since it is namespace-aware, it can have different configuration files for different XML applications. PassiveTEX is a large configuration file which maps XSL Formatting Objects onto a LATEX-based processing model in TEX, although very few of LATEX's front-end macros are visible. Another configuration file supplied with XMLTEX maps MathML onto TEX, allowing PassiveTEX to support XSL FO documents with embedded MathML.

How does XSL FO work? The language defines two primary objects: page masters, which define named styles of page layout; and page sequences, which reference a named page layout and contain a flow of text. Within that flow, text is assigned to one of five (rectangular) regions: the page body, areas at the top, bottom, left and right. We also have allowance for floating objects (at the top of the page), and footnotes (at the bottom), and the model covers writing in left/right and/or top/bottom modes. Within a region of text, we find one or more blocks, tables, lists and floats, while within a block (the equivalent to a TEX vertical box), we find inline sequences, characters, links, footnotes, and graphics. Associated with all these objects is an immense range of properties, divided into aural properties, borders, spacing and padding, breaking, colors, font properties (family, size, shape, weight, etc.), hyphenation, positioning, special table properties, and special list properties, although supporting absolutely all of them is not mandatory for a conforming processor.

It should be clear that the FO language should be able to describe the layout of most documents, by judicious combination of general purpose objects and their properties. The TEX user should note, however, that a FO document does not go as far as TEX in specifying exactly how pages will come out. It provides a set of constraints, but the exact line-breaking and page-breaking, for instance, can vary between implementations.

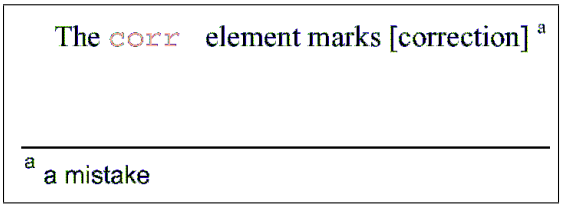
For an example of XSL FO, let us consider this piece of input XML written using the TEI (TEI Consortium, 2002) markup:

```
<p>The <gi>corr</gi> element marks
<corr sic="a mistake">correction</corr></p>
```

The <gi> markup says that the word 'corr' should be printed inside angle brackets, and the <corr> element should produce a footnote with the value of the 'sic' attribute. This text might be transformed into the following fragment of XSL FO:

```
<fo:block font-size="12pt"
 text-align="justify"
 text-indent="1em"
 space-before="0pt">
The <fo:inline
 color="green"
 font-family="Courier">corr
</fo:inline>
element marks [correction]
<fo:footnote>
<fo:inline font-size="8pt"
 vertical-align="super">a</fo:inline>
<fo:footnote-body>
<fo:block>
<fo:inline font-size="8pt"
 vertical-align="super">a</fo:inline>
<fo:inline
 font-family="Helvetica"
 font-size="10pt">a mistake</fo:inline>
</fo:block>
</fo:footnote-body>
</fo:footnote></fo:block>
```

and the result will look like this:



The PassiveTEX control file for XMLTEX which processes this XSL FO markup consists of a series of rules, one per element in the language. Each rule has three parts: a) any handling of attributes, b) what happens at the start of the element, and c) what happens at the end. This is demonstrated in the rule for floats:

```
\XMLelement{fo:float}
 {\XMLattributeX{float}{\FOfloat}{float}}
 {\ifx\FOfloat\att@none
     \begin{figure}[!htp]
   \else
     \begin{figure}
   \fi
   \FOlabel}
  {\end{figure}}
```

which does a fairly straightforward mapping of `<fo:float>` to LATEX's figure environment. A slightly more complex example is this rule for `<fo:inline>`:

```
\XMLelement{fo:inline}
 {}
 {\xmlgrab}
 {
 \ifx\FOverticalalign\att@auto
    \let\FOverticalalign\FObaselineshift
 \fi
 \FOlabel
 \ifx\FOborderstyle\att@solid
   \ifx\FOborderwidth\att@thin
      \def\FOborderwidth{0.4pt}
   \fi
   \ifx\FOborderwidth\att@medium
      \def\FOborderwidth{0.8pt}
   \fi
   \ifx\FOborderwidth\att@thick
      \def\FOborderwidth{1.2pt}
   \fi
   \FOboxedsequence{#1}%
 \else
      \FO@inlinesequence{#1}%
 \fi}
```

which shows some of the problems in mapping from word values for properties like 'medium'. The macros like `FO@inlinesequence` are defined in a large auxiliary file of helper macros for PassiveTEX.

Running PassiveTEX

Assuming you have created a file of XML using XSL FO vocabulary, you can use XMLTEX on a file called (say) `article.fo` in one of two ways:

1. Build an `xmltex` format file for pdfTEX with

   ```
   pdftex -ini "&pdflatex" pdfxmltex.ini
   ```

   and process your file with

   ```
   pdflatex "&pdfxmltex" article.fo
   ```

   Obviously you can create a command `pdfxmltex` to do this, which is just a script containing

   ```
   tex -fmt=pdfxmltex -progname=pdfxmltex
   ```

   Or,
2. Make a wrapper file called (say) `article.tex` along these lines:

   ```
   \def\xmlfile{article.fo}
   \input xmltex
   ```

   and run pdfTEX on it as normal with

   ```
   pdflatex article.tex
   ```

Do not worry, XMLTEX knows how to find the PassiveTEX macros as it needs them.

For reference, the PassiveTEX package consists of the following files:

- The core XMLTEX configuration files for XSL FO XML:

  ```
  fotex.xmt
  fotex.sty
  ```
- Support for direct formatting of TEI XML with XMLTEX

  ```
  tei.xmt
  teixml.sty
  ```

- Some support files, shared with JadeTEX:

```
unicode.sty
ucharacters.sty
mlnames.sty
dummyels.sty
```

Note that TEX has a limit on the length of line it can read, and some .fo files you generate may cause TEX to die with an message about increasing buf size. If you get that, edit your `texmf.cnf` file, increase the size of `buf_size` (mine is 200000), and remake any format files.

LATEX package dependencies This setup assumes you have a decent modern TEX setup. The TEX Live 7 CD-ROM is up to date (see `http://www.tug.org/texlive/`). Table 1 lists the packages loaded in a typical run of PassiveTEX, with their version numbers where known.

Extensions As explained above PassiveTEX effectively interprets MathML natively (elements must use the MathML namespace), and also supports a `bookmark` element in the `fotex` namespace, used to make PDF bookmarks. Usage is like this:

```
<fotex:bookmark
 xmlns:fotex="http://www.tug.org/fotex"
 fotex-bookmark-level="2"
 fotex-bookmark-label="ID">
 text of bookmark
</fotex:bookmark>
```

Notes on conformance to the XSL specification

The following general limitations apply to most of the PassiveTEX implementation of XSL FO:

1. The 'px' unit is not recognised.
2. Absolute dimensions always work, but proportional ones are often not recognized.
3. The functions allowed in attribute values are usually not recognized.
4. There is no error checking at all, and although all properties are recognized, do not assume that they do anything!

Most of the formatting objects are implemented more or less, except for the following:

1. `fo:bidi-override`
2. `fo:color-profile`
3. `fo:declarations`
4. `fo:initial-property-set`
5. `fo:instream-foreign-object`
6. `fo:multi-case`
7. `fo:multi-properties`
8. `fo:multi-property-set`
9. `fo:multi-switch`
10. `fo:multi-toggle`
11. `fo:region-end`
12. `fo:region-start`
13. `fo:table-footer`

The coverage of the myriad properties and valid values listed in the XSL FO specification is variable. All those that are straightforward to implement have been done; some are simply not relevant in TEX (e.g., the aural properties); some are just plain hard (repeatable column and rows in tables); others need help from (for example) Omega (bi-directional text). In some cases the TEX model just does not seem to fit—FO tables, for instance, work on the basis of cell properties, rather than TEX's idea of thinking about columns.

Tables are (unsurprisingly) the weakest area of PassiveTEX. Where column widths are specified, it does a reasonable job, but it has as yet no system for deriving column widths from data, as required by XSL FO. This is because TEX's table model has been abandoned in favour of the simple hbox and vbox constructs which can handle the endless variations on padding, borders and spacing.

Lastly, it should be noted the XSL FO inherits properties from Cascading Style Sheeets. CSS has a system of short-hands and composite values ("Times 12pt bold") which is painful to parse in TEX, and thus are largely not supported in PassiveTEX.

Things for LATEX users to remember

- No use is made of LATEX high-level constructs. No sections, no lists, no cross-references, no bibliographies; on the other hand, some extensions in the `fotex:` namespace have been implemented (for example, to get Acrobat bookmarks).
- XSL FO's underlying character set is Unicode; by default, entities are mapped to their Unicode position.
- All vertical and horizontal space is explicit in the specification.
- Page and line breaking is left to TEX: the rest is up to you.

References

[1] TEI Consortium, *Guidelines for Electronic Text Encoding and Interchange (TEI P4)*. Ed. C. M. Sperberg-McQueen and Lou Burnard. Chicago, Oxford: Text Encoding Initiative, 2002.

Table 1: LaTeX packages needed by PassiveTEX

amsbsy.sty	1999/11/29 v1.2d
amsfonts.sty	1997/09/17 v2.2e
amsgen.sty	1999/11/30 v2.0
amsmath.sty	2000/03/29 v2.08 AMS math features
amsopn.sty	1999/12/14 v2.01 operator names
amssymb.sty	1996/11/03 v2.2b
amstext.sty	1999/11/15 v2.0
array.sty	1998/05/13 v2.3m Tabular extension package (FMi)
article.cls	1999/09/10 v1.4a Standard LaTeX document class
bm.sty	1999/07/05 v1.0g Bold Symbol Support (DPC/FMi)
color.sty	1999/02/16 v1.0i Standard LaTeX Color (DPC)
fontenc.sty	(version not available)
graphics.sty	1999/02/16 v1.0l Standard LaTeX Graphics (DPC,SPQR)
graphicx.sty	1999/02/16 v1.0f Enhanced LaTeX Graphics (DPC,SPQR)
hpdftex.def	2000/05/08 v6.70f Hyperref driver for pdfTeX
hyperref.sty	2000/05/08 v6.70f Hypertext links for LaTeX
ifthen.sty	1999/09/10 v1.1b Standard LaTeX ifthen package (DPC)
keyval.sty	1999/03/16 v1.13 key=value parser (DPC)
longtable.sty	1998/05/13 v4.09 Multi-page Table package (DPC)
multicol.sty	1999/10/21 v1.5w multicolumn formatting (FMi)
nameref.sty	2000/05/08 v2.18 Cross-referencing by name of section
ot1phv.fd	2000/01/12 PSNFSS-v8.1 scalable font definitions for OT1/phv.
pd1enc.def	2000/05/08 v6.70f Hyperref: PDFDocEncoding definition (HO)
pifont.sty	2000/01/12 PSNFSS-v8.1 Pi font support (SPQR)
rotating.sty	1997/09/26, v2.13 Rotation package
size10.clo	1999/09/10 v1.4a Standard LaTeX file (size option)
stmaryrd.sty	1994/03/03 St Mary's Road symbol package
t1enc.def	1999/12/08 v1.9x Standard LaTeX file
t1phv.fd	2000/01/12 PSNFSS-v8.1 scalable font definitions for T1/phv.
t1ptm.fd	2000/01/12 PSNFSS-v8.1 font definitions for T1/ptm.
t2acmr.fd	1999/01/07 v1.0 Computer Modern Cyrillic font definitions
t2aenc.def	1999/11/29 v1.0c Cyrillic encoding definition file
t3enc.def	(version not available)
textcomp.sty	1999/12/08 v1.9x Standard LaTeX package
times.sty	2000/01/12 PSNFSS-v8.1 Times font as default roman (SPQR)
tipa.sty	1996/06/10 TIPA version 1.0
trig.sty	1999/03/16 v1.09 sin cos tan (DPC)
ts1cmr.fd	1999/05/25 v2.5h Standard LaTeX font definitions
ts1enc.def	1998/06/12 v3.0d (jk/car/fm) Standard LaTeX file
ts1ptm.fd	2000/01/12 PSNFSS-v8.1 font definitions for TS1/ptm.
ulem.sty	1997/04/21
umsa.fd	1995/01/05 v2.2e AMS font definitions
umsb.fd	1995/01/05 v2.2e AMS font definitions
upsy.fd	2000/01/12 PSNFSS-v8.1 font definitions for U/psy.
upzd.fd	2000/01/12 PSNFSS-v8.1 font definitions for U/pzd.
url.sty	1999/03/28 ver 1.5x Verb mode for urls, etc.
Ustmry.fd	(version not available)
uwasy.fd	1999/05/13 v1.0iWasy-2 symbol font definitions
wasysym.sty	1999/05/13 v1.0i Wasy-2 symbol support package

Indic Typesetting – Challenges and Opportunities

S. Rajkumar
Linuxense Information Systems, "Lalita Mandir", 16/1623,
Jagathy, Trivandrum 695014, India
raj@linuxense.com

Abstract

Asia boasts a wide variety of scripts, most of which are complex from the perspective of a computer scientist or engineer. This is true in the case of Indic scripts which are classified in the realm of complex scripts. All of the Indic scripts require a special process to transform from the a Unicode text to the actual glyph metrics for TEX to process.

In this paper I will talk about the processing of Unicode text to produce high quality typeset material for Indic scripts using OpenType fonts. I will cover the OpenType standards for Indic scripts and other facilities OpenType provides for advanced typesetting.

Introduction

TEX has remained and continues to remain one of the best typesetting systems of the world. But with the passage of time, TEX has been used for purposes that were not taken into account during its design phase. One such "flaw" is that it was based on 8-bit tables internally. This applies among others to the number of glyph in font files and the number of characters in a language. The original 8-bit design is fine for Roman scripts but inadequate for complex scripts like Indic or CJK scripts.

When TEX was designed there were very few programs that could really do any meaningful "typesetting". Font standards were almost nonexistent. As time passes the non-TEX world is also trying to catch up with TEX. One such promising technology is OpenType. It is an industry standard font technology formed by the fusion of TrueType and Type 1. OpenType has many advanced features that were the exclusive domain of TEX so far and adds even more.

The rest of the paper dwells more on the two subjects covered above and takes a look at how they can be used to advance the capabilities of TEX.

OpenType and Internationalization

One of the most exciting internationalization development happening outside the TEX world is the development of OpenType technology [1]. OpenType is an amalgamation of earlier TrueType and Type 1 technologies and designed from ground up to support complex scripts like Indic and Arabic. It uses a full 16 bits and is based on Unicode, and thus supports 64k glyph in a single font.

OpenType also supports advanced typographical control such as ligatures, kerning, small caps etc, which were available in TEX for a long time, plus swash variants, contextual ligatures, old-style figures, multi-script baselines etc, which are not part of TEX. What sets apart OpenType from others that offer these features, including TEX, is that the renderer need not be aware of all the features available in the fonts. OpenType fonts are capable of giving this information to the rendering engine. This enables the type designer to let her imagination run loose without being hampered by the limits of the rendering engine. If it is present in the font it will be used by the renderer.

This is in contrast with the TEX, where a clear encoding of a font is required for TEX to work. This problem has been sorted out in Latin scripts where the possible numbers of glyph to render a text is finite and a robust encoding scheme is in place for font designers to follow. But not so in Indic scripts. In languages like Malayalam there are very many conjuncts or ligatures possible, and not all fonts have all of them. Unlike in Latin scripts, the ligatures are not an advanced typographic tool but an absolute requirement for legible reading.

The number and placement of glyphs for high quality typography in Malayalam is still being debated. Furthermore, the Malayalam script itself is divided into an original script and a reformed script, each with a slightly different set of glyphs and rules for vowel consonants formation.

Another problem with Indic computing in general is that there are very few high quality fonts available and thus reusing fonts is a priority. Since the entire non-TEX world is moving towards OpenType as a single font standard, more and more new fonts to appear will be based on OpenType. All this points to the fact that a possible Omega implementation that uses OpenType will be ideal for Indic languages implementation of TEX.

OpenType Rendering for Indic scripts

OpenType rendering is the process of converting the plain Unicode input into a set of glyph indexes in a font file so that the text can be rendered on a device such as a screen or printer. This is much more complex than it sounds, since the font itself contains meta-information in the form of various *features*. Features are script- and language-dependent and standard features are registered in the renderer. This enables font designer to use the registered features so that consistent high quality output is produced when rendered and renderers can be written without being tied to particular fonts.

The standards for Indic scripts are designed by Microsoft Corporation [2] and are publically available. The latest Microsoft rendering engines support most of the Indic scripts but not all. Yudit [3] is one free editor which also supports most Indic languages.

An Indic OpenType rendering engine processes text in different stages. These stages include:

1. analyzing the syllables
2. reordering characters
3. shaping (substituting) glyphs according to the instructions in the font
4. positioning glyphs

During the analysis stage the engine takes the stream of Unicode characters and finds the syllable boundaries. Once a syllable boundary is found it is analyzed to find the features that can be applied to that syllable, such as possibly combining to produce a distinct glyph. Next the base consonant is identified. All other elements are classified according to the base consonant like pre-base, post-base, etc. Next the components that appear in more than one side are split into component parts. At this point the syllable is reordered according to the appropriate rules of the language. The reordered part is passed to the glyph substitution part of the engine to obtain a reordered glyph string. The various contextual shape features are applied to this nominal glyph string to obtain the final output for rendering.

Registered Features for Indic Scripts

Registered features can be divided into language features which encode the linguistic rules; conjuncts and typographic forms, which are used for typographical substitution; and conjunct creation and positioning features which are used to position the various markings for vowel modifiers along the final glyph. The language features listed in their order of applications are:

Linguistic Rules

nukt This feature takes nominal (full) forms of consonants and produces *nukta* forms. All nukta forms must be based on an input context consisting of the full form of consonants. All consonants in a font must have an associated nukta form, and nukta forms must exist in the font for all glyphs with akhand forms as well.

akhn This feature creates an *akhand* ligature glyph from two consonants in nominal forms separated by a halant. The input context for the akhand feature always consists of the full form of the consonant.

rphf Applying this feature produces the reph glyph. If the first consonant of the cluster consists of the full form (Ra + Halant), this feature substitutes the combining-mark form of Reph. In addition, the glyph that represents the combining-mark form of Reph is repositioned in the glyph string: it is attached to the final base glyph of the consonant cluster. The input context for the Reph feature always consists of the full form of Ra + Halant.

blwf Applying this feature creates below-base forms of consonants. The input context for the 'below-base form' feature must always consist of the full form of the consonant + Halant. The feature 'below-base form' is applied to consonants having below-base forms and following the base consonant. The exception is vattu, which may appear below half forms as well as below the base glyph. The feature 'below-base form' will be applied to all such occurrences of Ra as well.

half Applying this feature produces so-called half forms: forms of consonants used in pre-base position. Half forms must exist for all consonants in the font, and half forms of nukta consonants and Akhand consonants also must exist. Use the halant form for consonants that do not have distinct shapes for half forms. This feature is not applied to the base glyph even if the syllable ends with a halant.

pstf Applying this feature creates post-base forms. Examples include Bengali and Oriya 'Ya' and Malayalam 'Ya' and 'Va'.

vatu Vattu variants are formed by combining consonants with the vattu mark. Vattu ligatures can be either half or full form, and fonts must contain both. The input context for the 'vattu variants' feature must always consist of a consonant (in full or half form) + vattu glyph.

Conjuncts and Typographic Forms The conjuncts and typographical features are not strictly required, but almost all fonts will have to use some of these to produce readable text.

pres Pre-base substitutions: this feature produces conjuncts with half forms, the type most common in Devanagari. This feature also produces the correct shape of I-Matra (in Devanagari and similar scripts) and also may take care of pre-base matra ligatures like Tamil 'elephant trunk' shape of AI-Matra.

blws Below-base substitutions: This feature produces conjuncts of the base glyph with below-base consonants. Any specific context-dependent forms of below-base consonants are handled here as well. Finally, this feature produces matra ligatures with the base consonants and below-base stress and tone marks.

abvs Above-base substitutions: This feature produces the correct typographic shape when an above-base matra forms a ligature with the base glyph. This feature also produces conjuncts of the base glyph or matra with Reph, ligatures and forms involving above-base vowel modifiers and above-base stress and tone marks.

psts Post-base substitutions: This feature produces ligatures of the base glyph with post-base forms of consonants. It also produces the correct typographic shape when a post-base matra forms a ligature with the base glyph and different forms of post-base vowel modifiers like visarga.

haln Halant form of consonants: This feature produces the halant form of the base glyph in syllables ending with a halant. This features also takes care of *chillaksharams* in Malayalam.

Positioning Features

blwm Below-base marks: This feature positions all below-base marks on the base glyph.

abvm Above-base marks: This feature positions all above-base marks on the base glyph or the post-base matra.

dist This feature covers all other positioning lookups defining various distances between glyphs, such as kerning between pre and post-base elements (like Visarga) and the base glyph.

Moving Ahead

Currently the Omega typesetting system does not support OpenType technology. But this is one of the planned features for Omega in future.

References

[1] Adobe Corporation. `http://www.adobe.com/type/opentype/main.html`

[2] Microsoft Corporation. `http://www.microsoft.com/typography/otfntdev/indicot/features.htm`

[3] Sinai, Gasper. `http://www.yudit.org/`

METAOBJ: Very High-Level Objects in METAPOST

Denis Roegel
LORIA
Campus scientifique
BP 239
54506 Vandœuvre-lès-Nancy cedex
FRANCE
roegel@loria.fr
http://www.loria.fr/~roegel/

Abstract

This paper presents the METAOBJ system and its features for the implementation of very high-level objects within METAPOST.

Introduction

In recent years, METAPOST has become a very interesting and powerful tool for graphics, especially in the context of TEX, for which it is tailored. With METAPOST, the user writes a program describing a drawing. As a full-fledged programming language is at hand, it is possible to automate a number of tasks, and make use of functions for recurring parts in drawings. Therefore, one notable strength of METAPOST is its control of the drawings and the ease with which it becomes possible to ensure homogeneity. METAPOST appears also extremely useful in applications where drawings are generated automatically.

However, most applications to date are using only very simple METAPOST features. The power of METAPOST appears to have not yet been fully explored. One of the ideas which seemed worth exploring was the manipulation of objects and functions on objects within METAPOST. This was the initial aim of METAOBJ, the system which is presented here. METAOBJ is a very large extension of METAPOST, and the most complete (though not complete) description can be found in "The METAOBJ tutorial and reference manual" (Roegel, 2001).

In this article, we will present briefly the usual low-level way of drawing within METAPOST, as well as its shortcomings. We will then describe a functional approach to drawing, then show how objects can be implemented. An example will follow, before a more general overview of METAOBJ.

Low-level drawing in METAPOST

Several kinds of low-level drawings are possible in METAPOST.

Discardable drawings First, there are drawings which can be immediately used and which require no memorization. We call these drawings "discardable". An obvious example is drawing a square with:

```
draw (0cm,0cm)--(1cm,0cm)--
     (1cm,1cm)--(0cm,1cm)--cycle;
```

Many squares can be drawn that way, but if the user wants two squares with the same sizes, he/she must make sure that the drawing function uses the same values.

Hence, discardable drawings are sufficient for simple tasks, but as soon as the application becomes complex, they exhibit many drawbacks.

Memorizable drawings After discardable drawings, we have memorizable drawings. In this case, functions (or macros) are introduced and the use of these functions makes it much easier to obtain homogeneous drawings and to ensure that certain conditions are met. Functions can be called with parameters and producing two identical drawings only affords giving identical values to certain parameters. For instance, a more elaborate version of the square can be obtained with the following function:

```
def draw_Square(expr p,l,a)=
  draw p--(p+l*dir(a))--
       (p+l*dir(a)+l*dir(a+90))--
       (p+l*dir(a+90))--cycle;
enddef;
```

We now have a way to reuse drawing instructions. The `draw_Square` macro could even be enriched and perform other tasks than merely drawing a square.

The `draw_Square` macro actually contains the definition of the square. However, this macro does too much in that it also draws the square. We could

define a macro only defining the square as a path, for instance as:

```
def Square(expr p,l,a)=
  (p--(p+l*dir(a))
    --(p+l*dir(a)+l*dir(a+90))
    --(p+l*dir(a+90))--cycle)
enddef;
```

In that case, since `Square` returns a closed path, we can do several things with it. We can draw it with

```
draw Square(origin,1cm,50);
```

or we can fill it:

```
fill Square(origin,1cm,50);
```

Other options are also available.

However, even though we have a nice encapsulation of a square, such a definition may not be convenient for other "objects." For instance, if we want to define a function for a square with a double frame, we would not be able to define it in the same way, because a double frame is not a METAPOST path. In order to draw such an object, a special function may need to be introduced, for instance:

```
def drawDoubleSquare(expr p,l,a)=
  draw Square(p,l,a);
  draw Square(p-.5mm*dir(a+45),l+1mm,a);
enddef;
```

The point is that different objects are treated differently, which makes it cumbersome. A simple square can be drawn with `draw`, but a double square uses a special function.

Moreover, there are other underlying problems in the two approaches we have described, namely that the object itself is difficult to grasp. Even though the content of a double square is the definition of the `drawSquare` macro, the automatic analysis or extraction of parts of the definition is not easy. In the case of a simple square, the path could be memorized easily and all its points could be obtained by standard METAPOST functions, but other objects would need other treatments. This would result in different and non-homogeneous handling.

One might argue that it is not necessary to go beyond drawing, but in fact, there are examples where parts of objects need to be accessed once they are drawn; for instance, to specify certain non-global alignments.

We will now describe how objects can be accessed in an homogeneous way.

A functional approach to drawing

Before describing the implementation of objects, we will give an abstract description of the main features of an object. An object o is a structure with a name (o), points (which have names), linear equations between points, paths, and several other features that will be described later.

Objects will be of different types (classes), and to each type will correspond certain functions. Not all functions make sense with all types of objects. However, there is a category of "standard" objects to which the main functions apply. For instance, some of the functions are linear transformations: scaling, rotating, slanting or reflecting. When an object o is given, it can be rotated 90 degrees counterclockwise with `rotateObj(o,90)`.

Some of the functions are general (static) and not attached to an object, but others may be attached to an object and can be viewed as methods of the object. For instance, every time an object is drawn (with `drawObj(o)`), it is in fact a method of o, or more precisely a method defined in o's class which is called.

The functional approach to drawing is interesting because functions can return objects. By default, `rotateObj(o,90)` only rotates o, but cannot be composed. This corresponds to the imperative object-oriented construction of an object. It is object-oriented because o is an object and because a method of o is called. It is imperative because the operation is self-contained. In order to use a functional approach, METAOBJ provides a functional variant of `rotateObj` named `rotate_Obj`. It is then possible to write

```
rotate_Obj(rotate_Obj(o,60),30)
```

and this will return the initial object rotated $30+60$ degrees.

New functions can be written and added to the standard library. These functions will then take their power from the reflexion capability of METAOBJ, namely the possibility for the function to explore the object to which it applies, making therefore virtually anything possible.

The structure of an object

The main novelty of METAOBJ is the fact that it is a system which keeps track of all of an object's features. It is possible to reflect on an object's structure, and it is actually also possible to have functions creating objects, and even classes of objects, given certain parameters. The possibilities are endless and have hardly been explored so far.

An object has a name and all its direct components form a tree of variables, all starting with the object name. There is unfortunately no easy way to traverse such a tree of variables in METAPOST, and there are therefore special variables which keep

Attribute	Type	Default	Description
pointlist_	string	""	list of points
pairlist_	string	""	list of pairs (non-movable points)
pointarraylist_	string	""	list of arrays of points
subarraylist_	string	""	list of arrays of subobjects
stringarraylist_	string	""	list of arrays of strings
colorarraylist_	string	""	list of arrays of colors
picturearraylist_	string	""	list of arrays of pictures
transformarraylist_	string	""	list of arrays of transforms
booleanarraylist_	string	""	list of arrays of booleans
numericarraylist_	string	""	list of arrays of numerics
pairarraylist_	string	""	list of arrays of pairs (non-movable points)
points_in_arrayslist_	string	""	enumeration of all (movable) points of all arrays (of movable points)
picturelist_	string	""	list of pictures
numericlist_	string	""	list of numerics
sublist_	string	""	list of subobjects
subobjties_	string array		subobj tying equations (1 string/subobject)
nsubobjties_	numeric	0	number of subobjties
code_	string	""	code of an object
extra_code_	string	""	extra code of an object
ctransform_	transform	identity	current transform of that object

Table 1: Standard attributes of an object

track of what is in an object. We call these variables the *attributes* of an object. The complete list of attributes is given in table 1.

```
ptre=100
ptre.numericlist_="dx,dy,nst"
ptre.nst=2
ptre.ctransform_=(0,0,1,0,0,1)
```

Figure 1: Object structure: general data

As shown in figure 1, an object has a unique identifying number. Here, it is 100. Different numbers correspond to different objects. In particular, subobjects will also have their own identifying number.

Each object can store numerical values in the string `numericlist_`. There are three here, `dx`, `dy`, and `nst` (figure 1), but only `nst` is defined. This is the number of subtrees (2 in this example).

`ctransform_` records the current transform of the object, and it is initially the identity.

An object also has points (figure 2), the list of which is given in the string `pointlist_` which is a standard attribute of the object. It has a list of subobject references (figure 3), the list of which is given as a string `sublist_`. Each subobject is actually only given as a string. The subobject is not literally part of the object (an object can be part of several other objects).

The `ptre` object belongs to the `PTree` class (this information is not part of the object, but can be obtained by an external table) and it contains four subobjects. Each subobject has a corresponding string: `conc` (conclusion), `subt` (subtrees), `lr` (left rule), `rr` (right rule). The value of the string (figure 3) was generated automatically by `newobjstring_`. The fancy names avoid name clashes with user-defined objects.

The structure of `ptre` was obtained with

```
showObj ptre;
```

but this does not show the structure of all subobjects. We could define a function showing recursively the whole structure of an object, but currently the user must call `showObj` on each subobject `_______zu`, etc.

It is possible to go to great depth in an object. Even when there are `picture`s, we can find what is inside using the `for ... within` construct.

Each subobject has an associated tying function. This function attaches a subobject to the main object and is used when linear transformations are

```
ptre.pointlist_="ne,nw,sw,se,n,s,e,w,c,
                 ine,inw,isw,ise,in,is,ie,iw,ic,ledge,redge,lstart,lend"
ptre.c=(0,287.17845)
ptre.n=(0,312.73784)
ptre.s=(0,261.61906)
ptre.e=(95.02467,287.17845)
ptre.w=(-95.02467,287.17845)
ptre.ne=(95.02467,312.73784)
ptre.se=(95.02467,261.61906)
ptre.nw=(-95.02467,312.73784)
ptre.sw=(-95.02467,261.61906)

ptre.ic=(0,283.46451)
ptre.in=(0,308.30998)
ptre.is=(0,258.61905)
ptre.ie=(98.02467,283.46451)
ptre.iw=(-98.02469,283.46451)
ptre.ine=(98.02467,308.30998)
ptre.ise=(98.02467,258.61905)
ptre.inw=(-98.02469,308.30998)
ptre.isw=(-98.02469,258.61905)

ptre.ledge=(-72.57094,258.61905)
ptre.redge=(92.90178,258.61905)
ptre.lstart=(-51.91089,292.35118)
ptre.lend=(65.57219,292.35118)
```

Figure 2: Object structure (cont'd): points

```
ptre.sublist_="subt,lr,rr,conc"

ptre.conc="_______zu"
ptre.subt="_______zh"
ptre.lr="_______zs"
ptre.rr="_______zt"

ptre.nsubobjties_=4

ptre.subobjties_1="vardef tie_function_@#(expr $)=q_1=obj(@#subt).ne;
                   transformObj(obj(@#subt))($);
                   @#ne-obj(@#subt).ne=(p_1-q_1) transformed $;enddef;"
ptre.subobjties_2="vardef tie_function_@#(expr $)=q_2=obj(@#lr).ne;
                   transformObj(obj(@#lr))($);
                   @#ne-obj(@#lr).ne=(p_1-q_2) transformed $;enddef;"
ptre.subobjties_3="vardef tie_function_@#(expr $)=q_3=obj(@#rr).ne;
                   transformObj(obj(@#rr))($);
                   @#ne-obj(@#rr).ne=(p_1-q_3) transformed $;enddef;"
ptre.subobjties_4="vardef tie_function_@#(expr $)=q_4=obj(@#conc).ne;
                   transformObj(obj(@#conc))($);
                   @#ne-obj(@#conc).ne=(p_1-q_4) transformed $;enddef;"
```

Figure 3: Object structure (cont'd): subobjects

```
ptre.code_="@#se-@#sw=@#ne-@#nw;xpart(@#se-@#ne)=0;ypart(@#se-@#sw)=0;
            @#n=.5[@#ne,@#nw];@#s=.5[@#se,@#sw];@#e=.5[@#ne,@#se];
            @#w=.5[@#nw,@#sw];@#c=.5[@#n,@#s];@#ine=@#ne;@#inw=@#nw;
            @#isw=@#sw;@#ise=@#se;@#in=@#n;@#is=@#s;@#ie=@#e;@#iw=@#w;
            @#ic=@#c;
            xpart(.5[obj(@#conc).ledge,obj(@#conc).redge])=
               .5[xpart(obj(@#subt).s)-62.07625,xpart(obj(@#subt).s)+55.40683];
            ypart(obj(@#subt).s-obj(@#conc).n)=5.66928;
            ypart(@#n-obj(@#subt).n)=0;ypart(obj(@#conc).s-@#s)=0;
            @#ledge=obj(@#conc).sw;@#redge=obj(@#conc).se;
            ypart(@#lstart)=ypart(@#lend)=ypart(obj(@#conc).n)+2.83464;
            xpart(@#lstart)=xpart(obj(@#subt).c)-62.07625+0;
            xpart(@#lend)=xpart(obj(@#subt).c)+55.40683+0;
            xpart(@#e)-xpart(obj(@#subt).e)=+0;
            xpart(obj(@#subt).w)-xpart(@#w)=+20.33072;
            @#lstart-(rdistl,0)=obj(@#lr).e;"
```

Figure 4: Object structure (cont'd): equations

applied to an object. Having four subobjects, we have therefore four such functions, `subobjties_1`, ..., `subobjties_4`. These functions are stored as strings.

And finally, all the equations defining the initial state of the object are stored in the `code_` string (figure 4). These equations are used whenever an object is reset.

An object can contain more information, but every time there is some variable, the name of this variable must also appear in some list, because this is the only way to achieve duplication (cloning). We can only know what is inside an object if we constantly keep track of it. This also explains why special functions should be used to define variables. "`pair`" would not be enough to record the name of a pair. Instead, "`ObjPoint`" should be used.

An object definition example

The simplest of all classes is the `EmptyBox`. An `EmptyBox` is an empty rectangle, normally with no frame. Its only purpose is to take some space. For instance, it is useful in order to change the spacing between leaves of a tree, when the spacings are not all identical. The constructor looks like:

```
vardef newEmptyBox@#(expr dx,dy)
     text options=
  ExecuteOptions(options);
  assignObj(@#,"EmptyBox");
  StandardInterface;
  ObjCode StandardEquations,
   "@#ise-@#isw=(" & decimal dx & ",0)",
   "@#ine-@#ise=(0," & decimal dy & ")";
enddef;
```

It is called with two dimensions, which are the sides of the rectangle. It should be noticed that the values of `dx` and `dy` can be negative and this can produce some special effects.

The name of the object created is the suffix represented here as `@#`. The creation of box `b` would be done with:

```
newEmptyBox.b(2cm,1cm);
```

The syntax is therefore similar to the one found in the `boxes.mp` package.

The constructor also exhibits some features related to the options mechanism. Every constructor can have options modifying its behavior. The options are given as the last parameters of the constructor and are used in a call to `ExecuteOptions`. Each object decides which option it honors and how. The `EmptyBox` doesn't have many options, but it is still possible to draw its frame with a different thickness or to fill the box. Therefore, the `drawEmptyBox` function is (with slight simplifications):

```
def drawEmptyBox(suffix n)=
  if show_empty_boxes:
    drawFramedOrFilledObject_(n);
  fi;
  drawMemorizedPaths_(n);
enddef;
```

This macro is simple: depending on the global variable `show_empty_boxes` (often used for debugging), the empty boxes are shown or not. If they are shown, they are either filled or merely drawn. The `drawFramedOrFilledObject_` takes care of the various cases. If they are filled, they are filled with a color that can be given as an option.

If the object is filled, the "bounding path" of the object is used, and it is given by the function `BpathEmptyBox`. Like the function `drawObj` which calls `drawEmptyBox`, the `BpathObj` function actually

calls `BpathEmptyBox`. Each object must declare its "bounding path" function. For `EmptyBox`, we have

```
def BpathEmptyBox(suffix n)=
  StandardBpath(n)
enddef;
```

The standard bounding path provided by the `StandardBpath` function is merely the path `n.inw -- n.isw -- n.ise -- n.ine --cycle`. This path uses the "inner interface" so that the drawing of the object does not depend on artificial changes to its bounding box.

Overview of METAOBJ classes

METAOBJ defines standard object classes, which can then be instantiated. The standard library includes:

- basic objects: these objects are not containers and appear therefore at the leaves of a structure hierarchy:
  - `EmptyBox`: a rectangle with a given size;
  - `HRazor` and `VRazor`: this is a degenerated `EmptyBox`;
  - `RandomBox`;
- basic containers: such objects can contain any other object, and have varying shapes for the frame:
  - `Box`: a (usually) rectangular frame;
  - `Polygon`: a polygonal frame, whose number of sides can be parameterized;
  - `Ellipse`: an elliptic frame, whose shape can be parameterized;
  - `Circle`: a special case of `Ellipse`;
  - `DBox`: a box, with a doubled frame;
  - `DEllipse`: an ellipse, but with a doubled frame;
- box alignment constructors:
  - `HBox`
  - `VBox`
- recursive objects and fractals:
  - `RecursiveBox`: a box, containing a box, containing a box, ...
  - `VonKochFlake`: a well-known fractal;
- trees:
  - `Tree`: general trees;
  - `PTree`: proof trees;
- matrices: `Matrix`

A complex example

Figure 5 shows a much more elaborate example created with METAOBJ. The code provided should be easy to follow. First, a box named a is created, with the text "a". This is the square box "a" within the big circle, but at that point it is located nowhere. The box is actually "floating". Similarly, two ellipses are created, named b and c. The latter appears differently because the constructor (`newEllipse`) was called with several options which change the ellipse. From a, b, and c we create a tree, specifying the color of the arcs (which are arrows by default). The name of the tree is t and it is then put inside a box named aa. The new box is given round corners. This boxed tree is floating, until `aa.c=origin;` is executed. This is similar to how boxes are positioned with John Hobby's `boxes` package.

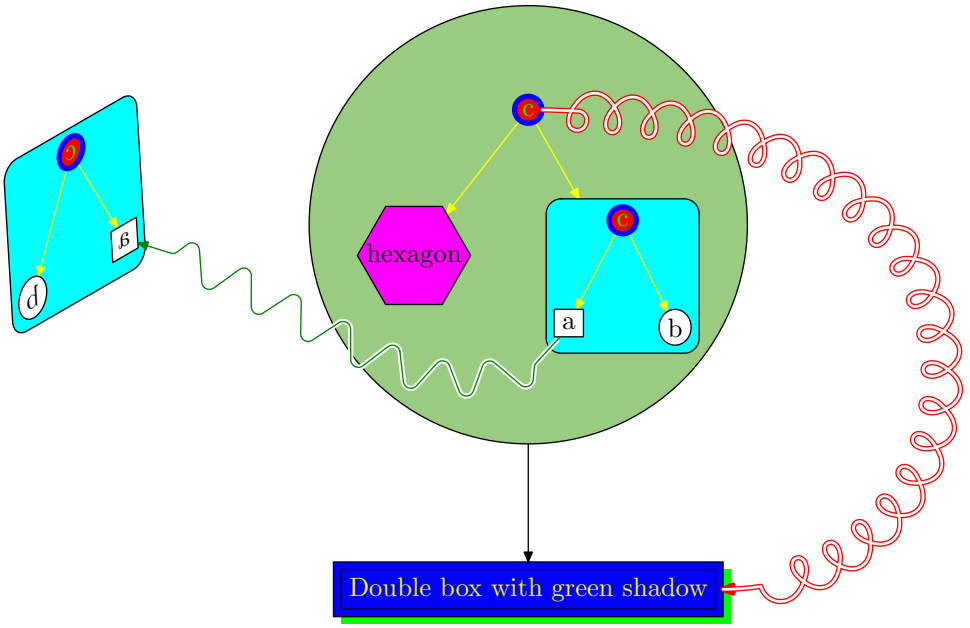
Later, another tree is built, and the whole tree is put inside a circle. This circle then becomes the root of a new tree and this tree is drawn with the call to `drawObj(nt)`. This automatically draws all components recursively. A spring then connects an ellipse (xc) to a double box (db). The spring (`nccoil`) has various options which were taken from PSTricks. It connects two named components without referring to the whole tree. However, it would have been possible to extract these two objects from the whole tree.

The rounded box (aa) is then duplicated and a new floating object dt is created. This copy is reflected, slanted and rotated. All these operations are applied recursively to all components of dt. The new object is drawn after having been put in place with `dt.c=nt.c-(6cm,-1cm)`. Finally, a zigzag connection is added between the former a object and its copy. The copy of a has an internal name, but we do not know it, and therefore we access the copy with the `treepos` function specifying the first child of dt.

Conclusion and related work

METAOBJ makes it possible to define and manipulate objects. The standard functions consider that the objects are rigid and can be combined in various ways. No matter how an object has been built, its structure is still easily accessible and open to introspection.

It is of course easy to add new objects and we have only provided a few. It is also easy for the programmer to write special packages manipulating special graphical formalisms, such as UML, etc.




```
input metaobj

beginfig(1);
  newBox.a("a");
  newEllipse.b("b");
  newEllipse.c("c") "filled(true)", "fillcolor(red)","picturecolor(green)",
                    "framecolor(blue)","framewidth(2pt)";
  newTree.t(c)(a,b) "linecolor((1,1,0))";
  newBox.aa(t) "filled(true)", "fillcolor((0,1,1))","rbox_radius(2mm)";
  aa.c=origin;
  newHexagon.xa("hexagon") "fit(false)","filled(true)","fillcolor((1,0,1))";
  newEllipse.xc("c") "filled(true)", "fillcolor(red)","picturecolor(green)",
                   "framecolor(blue)","framewidth(2pt)";
  newTree.xt(xc)(xa,aa) "linecolor((1,1,0))";
  newCircle.xaa(xt) "filled(true)", "fillcolor((.6,.8,.5))";
  newDBox.db(btex Double box with green shadow etex)
        "shadow(true)", "shadowcolor(green)",
        "filled(true)","fillcolor(blue)","picturecolor((1,1,0))";
  newTree.nt(xaa)(db);
  drawObj(nt);
  nccoil(xc)(db) "angleA(0)","angleB(180)",
    "coilwidth(5mm)","linetension(0.8)","linecolor(red)",
    "doubleline(true)","posB(e)";
  duplicateObj(dt,aa);
  reflectObj(dt,origin,up);
  slantObj(dt,.5);
  rotateObj(dt,30);
  dt.c=nt.c-(6cm,-1cm);
  drawObj(dt);
  nczigzag(a)(treepos(obj(dt.sub))(1))
    "angleA(-120)","coilwidth(7mm)","linecolor(.5green)","linearc(1mm)",
    "border(2pt)";
endfig;
```

Figure 5: A complex example and its source code.

It should also be possible to extend the concept of interface to other kinds of interfaces and to introduce objects that are not completely rigid. It is possible to use the complete object structure to implement tree layout algorithms. We could also have objects whose shape depends on parameters and on their location in a drawing. More elaborate operations could also be implemented if the history of an object was known. This is currently not the case, and causes limitations in certain experimental features such as the reset feature.

Other refinements include support for layers. This feature is currently not included in METAOBJ, but could be added in the future.

METAOBJ can be used as a replacement to the `boxes.mp`, `rboxes.mp`, and `fancybox.sty` packages and to many features found in PSTricks. Several features of METAOBJ, for instance connections, can be used without a reference to objects.

METAOBJ was especially influenced by my earlier work on animations (Roegel, 1997) where an object notion was introduced. The 3D objects were very primitive, but they provided many useful ideas. Several objects have been influenced from counterparts in various packages. This is especially true for rectangular and elliptic boxes. PSTricks provided many ideas and the connection functions are entirely borrowed from that package. Only PSTricks's user documentation was used, not the implementation details (van Zandt and Girou, 1994), though similarities can be observed. There has also been some work by Denis Girou on high-level objects in PSTricks (Girou, 1995), but this work is not really relevant to what we have done. Another work we didn't use was Kristoffer Rose's work on high level 2-dimensional graphics (Rose, 1997). The proof tree class was influenced by a LaTeX package I wrote in 1993 and which was never released.

Several systems explore similar ideas, though they did not influence METAOBJ. One is "Functional METAPOST," a system providing a Haskell layer to specify drawings abstractly (Korittky, 1998; Kuhlmann, 2001). The drawback of that approach is that one needs a Haskell compiler and of course one needs to learn some of this language. Nevertheless, this work shares many ideas with METAOBJ in that objects are built by applying functions to already existing structures. Another functional approach to picture drawing is FPIC which uses the ML language (Kamin and Hyatt, 1997). A very popular system with many similar ideas is the 2D Java API (Knudsen, 1999). It provides graphical objects which can be manipulated by various transformations and modified in various ways.

The syntax of METAOBJ is admittedly verbose, but it is hoped that a TeX interface will be provided and that it will alleviate a lot of the user's burden.

References

Girou, Denis. "Building high level objects in PSTricks". 1995. Slides presented at TUG'95, St. Petersburg (Florida).

Goossens, Michel, S. Rahtz, and F. Mittelbach. *The LaTeX Graphics Companion: Illustrating documents with TeX and PostScript*. Reading, MA: Addison-Wesley, 1997.

Hagen, Hans. *ConTeXt: the manual*, 2001a.

Hagen, Hans. *MetaFun*, 2001b.

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

Hoenig, Alan. *TeX Unbound: LaTeX & TeX Strategies for Fonts, Graphics, & More*. New York: Oxford University Press, 1998.

Kamin, Samuel N. and D. Hyatt. "A Special-Purpose Language for Picture-Drawing". In *USENIX Conf. on Domain-specific Languages, Santa Barbara*, pages 297–310. 1997.

Knudsen, Jonathan. *Java 2D Graphics*. O'Reilly & Associates, 1999.

Knuth, Donald E. *The METAFONTbook*. Reading, MA: Addison-Wesley, 1986.

Knuth, Donald E. *Digital Typography*, volume 78 of *CSLI Lecture Notes*. CSLI, 1999.

Korittky, Joachim. "Functional METAPOST. Eine Beschreibungssprache für Grafiken". 1998. Diplomarbeit an der Rheinischen Friedrich-WilhelmsUniversität Bonn.

Kuhlmann, Marco. "Functional METAPOST for LaTeX". 2001.

Ohl, Thorsten. "EMP: Encapsulated METAPOST for LaTeX". 1997. Technische Hochschule Darmstadt.

Roegel, Denis. "Creating 3D animations with METAPOST". *TUGboat* **18**(4), 274–283, 1997.

Roegel, Denis. *The METAOBJ tutorial and reference manual*, 2001.

Rose, Kristoffer Høgsbro. ""Very High Level 2-dimensional Graphics" with TeX and XY-pic". *TUGboat* **18**(3), 151–158, 1997.

van Zandt, Timothy. *PSTree user's guide*, 1993a.

van Zandt, Timothy. *PSTricks: PostScript macros for Generic TeX; User's Guide*, 1993b.

van Zandt, Timothy and D. Girou. "Inside PSTricks". *TUGboat* **15**(3), 239–248, 1994.

Typesetting in Hindi, Sanskrit and Persian: A Beginner's Perspective

Wagish Shukla
Maths Department
Indian Institute of Technology
New Delhi, India
wagishs@maths.iitd.ernet.in

Amitabh Trehan
Mahatma Gandhi Antarrashtriya Hindi Vishwavidyalaya (MGAHV)
16, 2nd floor, Siri Fort Road
New Delhi, India
amitabhtrehan@yahoo.co.in

Abstract

This paper describes our efforts to produce what is, to our knowledge, the first book typeset totally in an Indian language using LaTeX: *Chhand Chhand par Kumkum*, published by Prabhat Prakashan for Mahatma Gandhi Antarrashtriya Hindi Vishwavidyalaya (MGAHV).

We used the devnag package, which made it possible to encode each chapter, including verses, within a single set of \dn commands (much like an environment). Since then, we have also tried the sanskrit and ArabTeX packages and describe some of our experiences. Using devnag alone, typesetting a large file (a full-sized book) was a stable procedure. On the other hand, when using devnag and sanskrit together, even a small file can present problems. Using devnag/sanskrit in conjunction with ArabTeX is also problematic.

Additionally, one large part of the text was used to test conversion to HTML via latex2html (l2h) which has led to substantial upgrades of l2h by Ross Moore, its maintainer. This exemplifies the advantages of the free software community we have begun to live in. Ultimately, l2h was used to typeset MGAHV's website (http://www.hindivishwa.nic.in).

The Beginning

Our tryst with TeX began around the beginning of the year 2000 A.D. Since TeX/LaTeX is the best software for writing mathematical reports and we were in the mathematics department, we had come across mention of it here and there. Later we found that there were a few serious users, but most used GUI variants such as PCTeX (and not quite the latest ones!). The previous year the department and the institute had made rapid progress in computerisation and Internet connectivity, so every member of the faculty had a computer in his/her office and everybody (faculty and students) had round-the-clock Internet access. This prompted Wagish to think of what to do with the box in his office. He had previously stayed away from it religiously, but now he didn't want a relic in his room. So he decided to get 'computerised' and that's where Amitabh, who had recently started working with him as a student and welcomed the connectivity, came in handy. We picked up a lot of new ideas from the net, the airwaves and the brain waves and went about trying a few of them. Ultimately, we would have to say the most attractive ideas for us have been TeX, GNU/Linux and the free software philosophy.

Our first experiments using MikTeX, Ghostview, etc. to view mathematics papers were with Windows98 on a Pentium-II IBM machine (4GB HDD). Later, another computer (Pentium-III 500MHZ, 27GB HDD) and a laser printer were installed at the residence of Wagish Shukla and much of our work shifted there. We put up Redhat GNU/Linux and later Debian GNU/Linux on that machine. Meanwhile, TeXLive4.0, tugIndia, the tugIndia mailing list, CVR (C.V. Radhakrishnan) and like friends came along and we could do something useful.

The devnag Experience

Wagish writes in Hindi and needs to quote extensively from Sanskrit, Farsi and English, so it was natural that we should seek suitable solutions using LaTeX. Scanning the TeXLive4.0 package list, we came across the sanskrit, devnag and Indica packages. We couldn't find sanskrit and found no documentation for Indica. Fortunately, devnag was available, well documented and seemed friendly (important points for beginners). However, devnag on TeXLive4 was outdated (and still is, as of TeXLive6), making us suspect that we were in a less visited part of the forest. So, we downloaded devnag (v2.0, which had been upgraded to LaTeX2ϵ) from CTAN and set about experimenting with it. From the outset, the idea was to be able to produce large texts in Devanagari from it. As we progressed, it seemed that the developers' idea must have been to use it for short passages of Devanagari texts within English text but we are happy to state that we have been able to use it to typeset a whole book.

[tuglist] devnag + Windvi = Crash While using devnag with the TeXLive system with the Windows O.S., we came across a very strange problem. The devnag example and the test files compiled fine, so we made a small file with just some Devanagari text. This compiled and previewed well. Then we added some size-changing commands to it. It compiled. But as soon as we tried to preview it using Windvi (v. 0.66-pre6), Windows either went into a spate of blue-screen exception fault errors and rebooted or just rebooted without any warning. We copied the same file onto GNU/Linux and after removing the Microsoft newlines, we had no problem with the file. This was very intriguing. This happened to any devnag file which used size-changing commands (`\small`, `\large`, etc.)! So we posted the message on the list with the subject that takes the name of this subsection. Judging from the responses, hardly anybody on the list was using Windows (or if they did, they didn't respond). The problem indeed sounded strange to whoever heard it. Nobody could suggest what was wrong. Later, we also had some problems printing English files with Windvi. In a bit of hurry, we turned our attention to GNU/Linux and moved on.

In one of the discussions on the mailing list, C.V. Radhakrishnan had written: "Franz Velthius' simple preprocessor can seldom blow up a Win32 system". This leads us to suspect that the problems may have been caused by a virus or an anti-virus (we had Norton AntiVirus 2000 by then). Recently, when we tried to repeat the experiment with the same O.S. on the same machine, with TeXLive5, Windvi 0.67 and Norton Antivirus 2002, we had no such problems.

The Book Various experiments and Devanagari articles later, we came to do something really exciting. Wagish is a creator of many unfinished symphonies. Regarding TeX, Donald Knuth has written that it inspired him to write more and even rewrite his previous works because he could see his work beautifully written. Similarly, the transformation of his ideas typeset into a beautiful form have spurred Wagish to write more. The story of the book *Chhand Chhand par KumKum* had begun long ago, but somehow the book never materialised. Enthused by the idea of writing in Devanagari in a beautiful manner using the ethically beautiful idea of free software, Wagish thought that if it could be demonstrated that the author's creativity could be simply and beautifully expressed using the TeX system, it would inspire many people in many ways.

Chhand Chhand par KumKum is actually a commentary by Wagish of the famous poem "Ram Ki Shakti Puja" by Suryakant Tripathi Nirala, a very important poem in Hindi literature and considered rather difficult to discuss. Wagish wrote the criticism for one part of it (around a third), which was published in an issue of MGAHV's Hindi language literary magazine *Bahuvachan*. Though the rest of the issue was in a separate font using a different system, this article was printed using LaTeX. Thus, this issue has two distinct parts derived from two distinct systems. The look of the devnag font met with general appreciation and we ourselves were impressed with the intuitive commands and immense power that LaTeX and devnag offered. After this, the next logical step was to write the entire book using LaTeX and devnag.

Once this idea was concretised with support from MGAHV and its Vice Chancellor Ashok Vajpeyi and the arrangements worked out, we set to work. The whole contents of the book were then recreated and typed online by Wagish in almost exactly a month. The section previously published was also totally revised. For the general layout of the book, we used fancyheadings for the headers and footers and layout for testing the layout. Of course, our constant companions were the LaTeX book [1] and the *LaTeX Companion* book [2]. Our book was then put into final shape with help from other members of MGAHV and LILA (MGAHVś Laboratory for Informatics in the Liberal Arts), along with the publishers. Actually, in this area, publishers here

still look at our LaTeX experiment more as an idle curiosity than anything really useful.

While working with devnag we came across some interesting situations, described in the next section.

Critique Working with the devnag package on GNU/ Linux has been a pleasant experience. Bedore are some of our observations:

- In one of our first long articles, we just input the source file as a single paragraph without any line breaks. This is, of course, not a good practice, as it takes away from the readability of the text. When we used the devnag preprocessor, we were greeted by a segmentation fault. This was undoubtedly due to the limit of the text read into the character array in the preprocessor.
- The most useful feature is the transliteration scheme used by Frans Velthius. The whole text is typed in English and then converted by the preprocessor to a form suitable for LaTeX to generate the final output. Since this is a phonetic-based scheme, it is easy to remember. Moreover, the ligature construction is very close to the actual phonetic construction.
- The most attractive feature in devnag, which also highlights the advantage of a Character User Interface (CUI) approach versus a Graphical User Interface (GUI) approach, is the ligature construction. devnag has a wide range of ligatures. There is also the choice of switching individual ligatures on and off, as well as a broad subdivision of Hindi and Sanskrit ligatures.
- Just after a new line (\\), if a word begins with "qa", the "qa" is not processed. Thus

  ```
  {\dn
    namaskaara\\ qaafa
    }
  ```

  yields

  नमस्कार
  क़ाफ़

- The preprocessor does not always handle the verbatim environment properly (although it is supposed to). Thus, the segment in the item above with verbatim would be written as:

  ```
  {\dn
  nm-kAr\\  *A'
  }
  ```

  since it has preprocessed the contents.

- We wanted to write the word जुर्अत 'jurat', which reads normally as जुरत. By trial and error we discovered the way to input this was `jua\0ta`.
- For underlining a Devanagari passage, it is better to use the ulem package rather than the usual `\underline` command.
- Additional symbols were generated by using diacritics, as in a forthcoming book on Ghalib being written by Wagish; characters have been generated by using TIPA, which works well with devnag. For example, there are five letters in the Persian/Urdu alphabet which are, in India, homophonically pronounced as 'za'/ज़, but although devnag supplies 'za'/ज़, the five different versions were reproduced as follows:
  1. `za`/ज़ for Arabic ZE.
  2. `\textsubbar{za}`/ज़̲ for Persian/ Urdu ZAAL.
  3. `\textsubdot{za}`/ज़̣ for Persian/ Urdu ZVAD.
  4. `\textsubumlaut{za}`/ज़̤ for Persian/ Urdu ZOE.
  5. `\sout{za}`/ज̶़ for Persian ZE.

  The first four are from TIPA, the fifth from ulem. Similarly, in Persian/Urdu ख़्वाब, the व is not pronounced but written; thus, the pronounciation is ख़ाब but one must write ख़्वाब — the devnag input for ख़ाब is `.khaaba` and that for ख़्वाब is `.khvaaba` but it was impossible to indicate the same pronounciation with two differently spelled words. Instead, this was achieved by ख़्वाब (`\textsubw{.khvaa}ba`), using a command from TIPA.
- The compability of many LaTeX packages such as TIPA with devnag is heartening. However, ArabTeX does not mix well and loading sanskrit with either ArabTeX or devnag creates problems. Ideally, one would like to load all three (ArabTeX, sanskrit, devnag) at the same time.

LaTeX2HTML and devnag

MGAHV, a new university dedicated to Indian languages, literature, etc. needed to establish a website. Due to the profile of the university, it was necessary to have a bilingual website. We analysed the available options and found that there really wasn't any standard solution for setting up a website in Devanagari. One important criteria for us was that our site should be accessible uniformly across platforms and browsers: that is, setting up the site with some specific font made available for download

was not an attractive option. Most sites that use this solution can only be accessed on the Windows platform after installing the proper font. Needless to say, in this age of viruses and worms, one is rather hesitant to install something to view a site. There is the option of using dynamic fonts but we were not sure about reliability, the degree of complexity of such a solution and whether there was anything in the free software domain for this. So, it seemed that we needed some image-based solution for our limited needs, but one which would not bloat up the size of the files, so that access remained reasonably fast. Given our devnag experience, we hoped to find something similar in nature. And we did — LaTeX2HTML (l2h), which also provided support for devnag.

Development via the net It was a bit of a bumpy ride getting l2h working for devnag: it turned out that nobody, to our knowledge, had used it before. Thus, like Wagish's book, MGAHV's site is also the first one created via this route. We attempted to run l2h on our devnag files and constantly mailed queries to the current maintainer, Ross Moore, who kept on advising and correcting bugs till, at last, l2h ran pretty well with devnag. This was, for us, a unique experience of software development via the Internet in the free software domain and highlighted the advantages and the cooperative spirit that this approach can generate.

l2h generates PNG/GIF images for things not directly available via HTML, such as mathematics and Indian language characters. This is where things get complicated, as l2h depends on the support of a number of other applications for image generation, including the netpbm suite of files. We installed l2h from source and then tried the package made by Manoj Srivastava for Debian on our Debian system, but the images wouldn't generate. So we joined the mailing list and realised that we needed to update netpbm. Once upgraded, the `"make test"` with l2h worked and everything seemed to be ready. But when we tested it with a small sample file it wouldn't work: it couldn't locate the devnag style files and generate images, even though it would work on Ross's system. We had also copied the l2h Indic-TEX devnagri.sty and devnagri.perl files to particular locations, as indicated in the l2h documentation. That's when Ross realised that the files for the upgraded devnag had not been uploaded for distribution. So he took care of that. By default the system had been set to use the DN2 preprocessor with devnag (DN2 is used with texts in German). Ross changed the default and left DN2 as an option.

Since we had now made some progress, we decided to give it a more thorough test. We fed l2h Wagish's article, "Ram Ki Shakti Puja", mentioned in the previous section — a file of 89Kb. l2h invokes LATEX to generate images, but it complained of memory shortage and halted. Moreover, the log indicated that l2h was trying to create just three images from the whole document. The cause of this problem turned out to be very interesting.

The article actually had a very typical structure which may not, however, have been envisioned by the developers. There were many verse environments within a single set of `\dn` braces whereas the developers had probably expected a set of `\dn` braces for each verse, so l2h was trying to generate huge images and collapsed. Ross improved the paragraph breaking, also adding an option for newlines within the title command and ultimately put up the converted document on his site. And so Lord Rama now adorns the net as a test case.

Satisfied with the results, we carried the experiment forward and created the LILA website (`www.hindivishwa.nic.in`). The images are set against a white background and the web document looks good. Overall, feedback about the quality and speed of access has been positive from people who have visited the site. The ultimate solution is probably going to come with the use of Unicode and like encodings, but we think that, with some more facilities, l2h would make a good substitute in the meanwhile.

Critique

- l2h has proven to be a good solution for sites with static Devanagari content. PNG images are of a reasonable size and don't slow down the site too much.
- At times, there are problems with clipping of the boxes around images.
- We need to have an easier update system (a sort of version control and patch system) for updating image-based sites. This is because it takes longer to process the whole text, even if one just wants to add, say, a page to the original. It would also be much easier to just upload/delete a few images instead of the whole site, which may be required for changes at the present. Thus, such a package could provide content additions, deletions and updating facilities.
- There is probably a need for closer collaboration between the developers of l2h and say, netpbm, to maintain compatibility.

devnag, sanskrit and ArabTEX

In the book on Ghalib (in Hindi) that Wagish is presently working on, we needed to use Arabic. So, we tried to use ArabTEX with sanskrit and devnag, in various permutations. The results were not very good:

- sanskrit(skt) and devnag, when taken together, make the typeset words look weird.
- ArabTEX and sanskrit or devnag output certain Greek letters at the beginning of a document and don't process the text properly.

While devnag has been useful to us, there are facilities in other packages which could be useful if incorporated into devnag:

- ArabTEX doesn't use a preprocessor
- sanskrit has support for vedic Sanskrit (but not Hindi)
- both ArabTEX and sanskrit incorporate standard transliteration facilities
- sanskrit has both bold and italic fonts

Conclusions

There is a need for more language-specific development on TEX systems, if publishers in Indian languages are to be convinced to start using TEX. Some improvements which could be made do not seem extremely difficult for the developers. There is also a need for greater variety in the form of fonts, etc. Native speakers of the language should get involved in at least the testing of suitable packages, as they could provide some unique insights.

References

[1] Leslie Lamport. *LATEX: A Document Preparation System*. Addison-Wesley, Reading, Massachusetts, second edition, 1994.

[2] Michel Goosens, Frank Mittelbach and Alexander Samari. *The LATEX Companion*. Addison-Wesley, Reading, Massachusetts,1994.

[3] Wagish Shukla. *Chhand Chhand par Kumkum*. Prabhat Prakashan, New Delhi, 2001.

New Typesetting Language and System Architecture

Karel Skoupý
Switzerland
skoupy@inf.ethz.ch

Abstract

The TEX input language is quite flexible for manipulation of the document's contents, but it is too limited for efficient work with its actual typesetting (layout). Use of a more general programming language was already proposed by Frank Mittelbach in 1990. Significant inconveniences of the existing TEX programming capabilities are also experienced in development of ConTEXt by Hans Hagen.

There are two kinds of problems. One is the oversimplified type system of TEX, which doesn't support structured and user defined types; also, the control structures might be richer, although this is probably not critical. The second problem is that the set of TEX primitives is not complete: for example, there is no `\lastrule` or `\lastspecial`. These problems usually lead to unreliable workarounds for complex programming tasks like multicolumn typesetting.

In my new project I plan to provide an alternative and more viable language for typographic programming. It will also support a unified model of text and graphics, an another part of the project.

The project also aims to provide a modular system architecture which separates the language and the typesetting engine. The architecture is intended to support multiple languages: TEX compatibility mode, Scheme with typographic primitives, ... This future typesetting system will be composed of flexible components which can support multiple input (TEX, XML) and output (DVI, PostScript, PDF) formats and different font types.

TEX File Server

Karel Skoupý
Switzerland
skoupy@inf.ethz.ch

Abstract

Repeated re-processing of TEX documents is a widely used technique for resolving references and generating properly placed graphics. In this process the data structures for efficient finding of files built by kpathsea are lost after each run.

We will discuss how this problem can be eliminated by developing a TEX file server. The added value will be cross-network transparency and resource sharing. In the paper we discuss the possible approaches, propose the server protocol and integration with kpathsea. The prototype of the server will be demonstrated.

Conserving Implicit Mathematical Semantics in Conversion between TEX and MathML

Stephen M. Watt
Ontario Research Center for Computer Algebra
University of Western Ontario, London, Canada N6A 5B7
Stephen.Watt@uwo.ca
http://www.csd.uwo.ca/~watt

Abstract

MathML[1] is an XML representation for mathematical objects, allowing expressions to be stored in databases, transmitted between applications and operated upon by programs. MathML can be used to express mathematical content in web pages and digital libraries, and has become an accepted form for input and output of computer algebra systems. There have been several efforts to design software for conversion of mathematical expressions from TEX to MathML and vice versa. We consider the problem of how conversion between formats can conserve any mathematical semantics implied by the markup of the original document.

Both TEX and MathML admit macro mechanisms, natively so in TEX and via XSLT[2] with MathML. Authors may use pre-defined style-sheets or define their own abbreviations, effectively extending the vocabulary of the environment. Macros are typically used as shorthands for lengthy expressions or to maintain notational independence. A simple example of notational independence would be, e.g., to define `\Vector` to expand either to `\mathbf{v}` or `\vec{v}` or something else, depending on the style sheet used.

Any serious converter between TEX and MathML must support macros. The standard approach has been to expand macros, and then perform the translation from low-level TEX to MathML (or vice versa). We see this approach as undesirable, as the use of macros in practice captures mathematical semantics within expressions. For example, if the form `\BesselJ{\alpha}{z}` expands to $J_\alpha(z)$, and translates to

```
<mrow>
    <msub><mi>J</mi><mi>&alpha;</mi></msub>
    <mfenced><mi>z</mi></mfenced>
</mrow>
```

then we have lost the fact that J is a Bessel function. The MathML-processing application will have no way to determine that this is a Bessel function, and not something else, e.g., an angular momentum or a jet bundle.

Our approach has been quite different: We assume that in many interesting cases it shall be possible to map macros in one setting to corresponding macros in another, thus conserving implied semantics. We have adopted the hypothesis that certain TEX style or class files will naturally have counterpart XSLT style-sheets. Particular TEX macros then correspond to specific XSLT template definitions. The conversion process in either direction must be able to recognize and use these correspondences. Additionally, we provide fine control over which macros are to be carried over to the target markup and which are uninteresting shorthands to be expanded.

This paper presents the overall architecture of a pair of programs — one converting mathematical TEX to MathML[3] and one converting MathML to TEX[4]. Both of these programs permit macros to be translated at a high semantic level. These may be specified individually, or as sets corresponding to style files. Other macros are expanded to primitive forms for translation.

Declarative mapping files are used to specify the correspondence between TEX and XML forms, typically for TEX macros and XSLT templates, and the same file may be used for conversion in either direction. Multiple mapping files may be used, if desired, corresponding to multiple macro definition files. We discuss how this architecture can be used with content, presentation or combined MathML markup, and how the programs can be used in conjunction with other tools for conversion between TEX and other XML formats.

— — ∗ — —

References

[1] Mathematical Markup Language (MathML) Version 2.0, D. Carlisle, P. Ion, N. Poppelier and R. Miner (eds), R. Ausbrooks, S. Buswell, S. Dalmas, S. Devitt, A. Diaz, R. Hunter, B. Smith, N. Soiffer, R. Sutor and S. Watt, World Wide Web Consortium Recommendation, 21 February 2001, http://www.w3.org/TR/2001/REC-MathML2-20010221.

[2] XML Transformations (XSLT) Version 1.0, J. Clark (ed), World Wide Web Consortium Recommendation, 16 November 1999, http://www.w3.org/TR/1999/REC-xslt-19991116.

[3] A TEX to MathML Converter, I. Rodionov and S. Watt, http://www.orcca.on.ca/MathML/texmml/textomml.html.

[4] A MathML to TEX Converter, E. Smirnova and S. Watt, http://www.orcca.on.ca/MathML/texmml/mmltotex.html.

Calendar

2002

Feb 20 – 23 DANTE 2002, 26th meeting, Universität Erlangen-Nürnberg, Germany. For information, visit http://www.dante.de/dante2002/.

Apr 29 – May 3 EuroBachoTEX 2002, 13th meeting of European TEX Users and 10th annual meeting of the Polish TEX Users' Group (GUST), "TEX and beyond", Bachotek, Brodnica Lake District, Poland. For information, visit http://www.gust.org.pl/BachoTeX/2002/.

May 29 Journée GUTenberg, "Distributions", Paris, France. For information, visit http://www.gutenberg.eu.org/.

TUG 2002
International Convention Centre, Trivandrum, India. For information, visit http://www.tug.org.in/tug2002/.

Sep 1 – 3 Tutorials: LATEX; LATEX to XML; METAPOST; the Text Encoding Initiative; TEX macro expansion.

Sep 4 – 7 The 23rd annual meeting of the TEX Users Group, "Stand up and be proud of TEX!". For information, visit http://www.tug.org.in/tug2002/.

Oct 12 UK TUG Autumn meeting, Nottingham University. For information, contact Dick Nickalls, dicknickalls@compuserve.com.

2003

Mar 24 – 28 IUC23, The 23rd Internationalization and Unicode Conference, "Unicode, Internationalization, the Web: The Global Connection", Prague, Czech Republic. For information, visit http://www.unicode.org/iuc/iuc23/.

Apr 2 – 4 DANTE 2003, 28th meeting, Universität Bremen, Germany. For information, visit http://www.dante.de/dante2003/.

May 1 – 3 BachoTEX 2003, 11th annual meeting of the Polish TEX Users' Group (GUST), Bachotek, Brodnica Lake District, Poland. For information, visit http://www.gust.org.pl/BachoTeX/2003/.

May 15 – 17 Typo Berlin 2003, the 8th International Design Conference, Berlin, Germany. For information, visit http://www.typo-berlin.de.

May 28 – 30 Society for Scholarly Publishing, 25th annual meeting, "Navigating Change", Baltimore, Maryland. For information, visit http://www.sspnet.org.

May 29 – Jun 2 ACH/ALLC 2003: Joint International Conference of the Association for Computers and the Humanities, and Association for Literary and Linguistic Computing, "Web X: A Decade of the World Wide Web", University of Georgia, Athens, Georgia. For information, visit http://www.english.uga.edu/webx/ or the organization web site at http://www.ach.org.

Jun 11 – 13 Seybold Seminars PDF Summit, Amsterdam, Netherlands. For information, visit http://www.seyboldseminars.com/pdf_summit/.

Jun 12 NTG 31st meeting; no details yet.

Jun 24 – 27 EuroTEX 2003, "Back to Typography", Brest (Brittany), France. For information, visit http://omega.enstb.org/eurotex2003/. (The EuroTEX 2003 proceedings will be published in *TUGboat*.)

Status as of 1 February 2003

For additional information on TUG-sponsored events listed here, contact the TUG office (+1 503 223-9994, fax: +1 503 223-3960, e-mail: office@tug.org). For events sponsored by other organizations, please use the contact address provided.

Additional type-related events are listed in the Typophile calendar, at
http://www.icalx.com/html/typophile/month.php?cal=Typophile.

Owing to the lateness of this issue, please consider that all events shown for 2002 are included only "for the record".

Jul 7 – Aug 8 — Rare Book School Summer Session, University of Virginia, Charlottesville, Virginia. A series of one-week courses on topics concerning rare books, manuscripts, the history of books and printing, and special collections. For information, visit `http://www.virginia.edu/oldbooks`.

TUG 2003
Outrigger Waikoloa Beach Resort,
Big Island, Hawai'i.

Jul 15 – 18 — Beginning/Intermediate LaTeX, at the University of Hawaii at Hilo. For information, visit `http://www.tug.org/tug2003/latexclass.html`.

Jul 20 – 24 — The 24th annual meeting of the TeX Users Group, "Silver Anniversary — 25 years! — of TeX". For information, visit `http://www.tug.org/tug2003/`.

Jul 17 – 20 — TypeCon2003, "Counter Culture", Minneapolis, Minnesota. For information, visit `http://www.typecon2003.com/`.

Jul 27 – Aug 1 — SIGGRAPH 2003, San Diego, California. For information, visit `http://www.siggraph.org/calendar/`.

Aug 3 — Web Document Analysis workshop, Edinburgh, Scotland, UK. For information, visit `http://www.csc.liv.ac.uk/~wda2003`.

Aug 3 – 6 — ICDAR 2003, International Conference on Document Analysis and Recognition, Edinburgh, Scotland, UK. For information, visit `http://www.essex.ac.uk/ese/icdar2003/`.

Sep 8 – 9 — DANTE 29th meeting, Universität Giessen, Germany. For information, visit `http://www.dante.de/events/mv29/`.

Sep 22 – 25 — Seybold San Francisco, San Francisco, California. For information, visit `http://www.seyboldseminars.com/sf2003/`.

Sep 19 – 22 — Association Typographique Internationale (ATypI) annual conference, "Between Text and Reader", Vancouver, Canada. For information, visit `http://www.atypi.org/40_conferences`.

Oct 20 – 21 — Second Annual St. Bride Conference, London, England. For information, visit `http://www.stbride.org/calendar.htm`.

Dec 7 – 12 — XML 2003, Philadelphia, Pennsylvania. For information, visit `http://www.idealliance.org/events_upcoming.asp`.

Institutional Members

American Mathematical Society, *Providence*, *Rhode Island*

Banca d'Italia, *Roma*, *Italy*

Center for Computing Science, *Bowie*, *Maryland*

Cessna Aircraft Company, *Wichita*, *Kansas*

The Clarinda Company, *Clarinda*, *Iowa*

CNRS - IDRIS, *Orsay*, *France*

CSTUG, *Praha*, *Czech Republic*

Florida State University, School of Computational Science and Information Technology, *Tallahassee*, *Florida*

IBM Corporation, T J Watson Research Center, *Yorktown*, *New York*

Institute for Advanced Study, *Princeton*, *New Jersey*

Institute for Defense Analyses, Center for Communications Research, *Princeton*, *New Jersey*

Kluwer Academic Publishers, *Dordrecht*, *The Netherlands*

KTH Royal Institute of Technology, *Stockholm*, *Sweden*

Masaryk University, Faculty of Informatics, *Brno*, *Czechoslovakia*

Max Planck Institut für Mathematik, *Bonn*, *Germany*

New York University, Academic Computing Facility, *New York*, *New York*

Princeton University, Department of Mathematics, *Princeton*, *New Jersey*

Siemens Corporate Research, *Princeton*, *New Jersey*

Springer-Verlag Heidelberg, *Heidelberg*, *Germany*

Stanford Linear Accelerator Center (SLAC), *Stanford*, *California*

Stanford University, Computer Science Department, *Stanford*, *California*

Stockholm University, Department of Mathematics, *Stockholm*, *Sweden*

University College, Cork, Computer Centre, *Cork*, *Ireland*

University of Delaware, Computing and Network Services, *Newark*, *Delaware*
Ste-Foy, Québec, Canada¡

University of Oslo, Institute of Informatics, *Blindern*, *Oslo*, *Norway*

Vanderbilt University, *Nashville*, *Tennessee*

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

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.

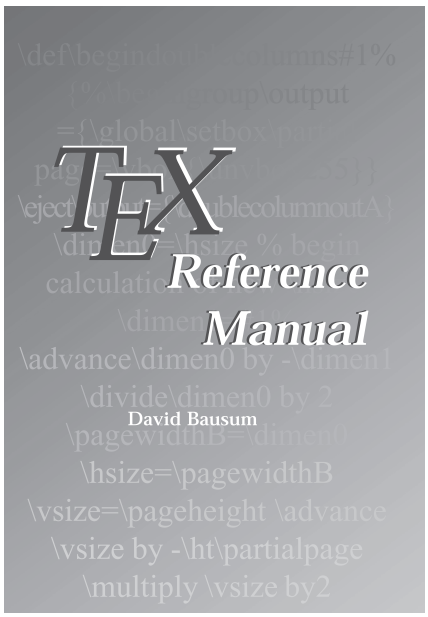
Ogawa, Arthur
40453 Cherokee Oaks Drive,
Three Rivers, CA 93271-9743;
(209) 561-4585
Email: Ogawa@teleport.com
Bookbuilding services, including design, copyedit, art, and composition; color is my speciality. Custom TEX macros and LATEX2ε document classes and packages. Instruction, support, and consultation for workgroups and authors. Application development in LATEX, TEX, SGML, PostScript, Java, and ßC++. Database and corporate publishing. Extensive references.

Veytsman, Boris
2239 Double Eagle Ct.
Reston, VA 20191;
(703) 860-0013
Email: boris@lk.net
I provide training, consulting, software design and implementation for Unix, Perl, SQL, TEX, and LATEX. I have authored several popular packages for LATEX and `latelx2html`. I have contributed to several web-based projects for generating and typesetting reports.
For more information please visit my web page: http://users.lk.net/ borisv.

The Unicorn Collaborative, Inc, Ted Zajdel
115 Aspen Drive, Suite K
Pacheco, CA 94553
(925) 689-7442
Email: contact@unicorn-collab.com
We are a technical documentation company, initiated in 1990, which time, strives for error free, seamless documentation, delivered on time, and within budget. We provide high quality documentation services such as document design, graphic design and copy editing. We have extensive experience using tools such as FrameMaker, TEX, LATEX, Word, Acrobat, and many graphics programs. One of our specialties is producing technical manuals and books using LATEX and TEX. Our experienced staff can be trained to use any tool required to meet your needs. We can help you develop, rewrite, or simply copy-edit your documentation. Our broad experience with different industries allows us to handle many types of documentation including, but not limited to, software and hardware systems, communications, scientific instrumentation, engineering, physics, astronomy, chemistry, pharmaceuticals, biotechnology, semiconductor technology, manufacturing and control systems. For more information see our web page http://www.unicorn-collab.com.

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.



TEX *Reference Manual*

by **David Bausum,** *Lighthouse & Associates, Beloit, WI, USA*

The *TeX Reference Manual* is the first comprehensive reference manual written by a programmer for programmers. It contains reference pages for each of TeX's 325 primitive control sequences. Over 80% of its reference pages contain examples that range from simple to challenging. Each example is is typeset verbatim in a style which is easy to read and experiment with. *TeX Reference Manual* also just typesets the example, so you can see what it makes, and explains how the example works. The description on each primitive's reference page is an annotated discussion of *The TeXbook's* treatment of the primitive. That means a TeX user will find it natural to move back and forth between the two books. One of *TeX Reference Manual's* innovative features is families. They simplify the search for the primitive which performs a particular task.

Primitive Control Sequences

Family Name		Type	Description
Box (29)	Logic (20)	C	Command (163)
Character (16)	Macro (20)	D	Derived Command (17)
Debugging (25)	Marks (4)	IQ	Internal Quantity (42)
File I/O (13)	Math (69)	PI	Parameter (integer) (55)
Fonts (5)	Page (13)	PD	Parameter (dimen) (21)
Glue (12)	Paragraph (30)	PG	Parameter (glue) (15)
Hyphenation (11)	Penalties (12)	PM	Parameter (muglue) (3)
Inserts (8)	Registers (11)	PT	Parameter (token) (9)
Job (11)	Tables (9)		
Kern (7)			

TeX Reference Manual has appendices which provide a comprehensive discussion of: verbatim material, PostScript fonts, and two-column material. In particular, one word describes its font macros, elegant. The *TeX Reference Manual* is an invaluable tool for both the experienced and new users of TeX.

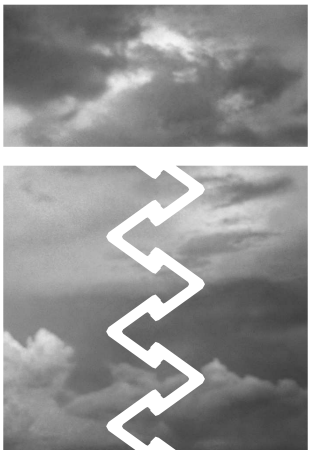
CONTENTS

February 2002 Hardbound, ISBN 0-7923-7673-0 390 pp.
EUR 108.00 / USD 99.00 / GBP 68.00
Special Price offered to TUGBOAT subscribers:
EUR 97.00 / USD 90.00 / GBP 61.00

introducing

TEXTURES® 2.0

WITH SYNCHRONICITY



AGAIN THE MACINTOSH DELIVERS A NEW TEX WITH A REVOLUTION IN HUMAN INTERFACE.

As computer power has advanced, the Macintosh has consistently been the leader in the human and humane connection to technology, and Textures has consistently led in bringing ease of use to TEX users. First with Textures 1.0, the first truly integrated TEX system. Then with Lightning Textures, the first truly interactive TEX system. Now, with Textures 2.0 and Synchronicity, Blue Sky Research again delivers a striking advance in TEX interactivity and productivity.

With Synchronicity, your TEX input documents are reliably and automatically cross-linked, correlated, or "synchronized" with the finished TEX typeset pages. Every piece of the finished product is tied directly to the source code from which it was generated, and vice-versa. To go from TEX input directly and exactly to the corresponding typeset characters, just click. It's that simple: just click, and Textures will take you instantly and precisely to the corresponding location. And it goes both ways: just click on any typeset character, and Textures will take you directly to the TEX code that produced it. No matter how many input files you have, no matter what macros you use, Synchronicity will take you there, instantly and dependably.

Improve YOUR *performance:*

GET SYNCHRONICITY

BLUE SKY RESEARCH
317 SW ALDER STREET
PORTLAND, OR 97204 USA

800 622 8398
503 222 9571
WWW.BLUESKY.COM