TUGboat

Volume 27, Number 1 / 2006
EuroTEX 2006 Conference Proceedings

TeX Users Group

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

Memberships and Subscriptions
2006 dues for individual members are as follows:
- Ordinary members: $85.
- Students/Seniors: $45.

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

Membership in the TeX Users Group is for the calendar year, and includes all issues of *TUGboat* for the year in which membership begins or is renewed, as well as software distributions and other benefits. Individual membership is open only to named individuals, and carries with it such rights and responsibilities as voting in TUG elections. For membership information, visit the TUG web site.

Also, (non-voting) *TUGboat* subscriptions are available to organizations and others wishing to receive *TUGboat* in a name other than that of an individual. The subscription rate is $95 per year, including air mail delivery.

Institutional Membership
Institutional membership is a means of showing continuing interest in and support for both TeX and the TeX Users Group, as well as providing a discounted group rate and other benefits. For further information, contact the TUG office or see our web site.

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
+1 206 203-3960

Electronic Mail
(Internet)

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

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

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

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

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

Have a suggestion? Problems not resolved?
The TUG Board wants to hear from you:
Please email `board@tug.org`.

[printing date: October 2006]

TUGBOAT

The Communications of the TEX Users Group

Volume 27, Number 1, 2006
EuroTEX 2006 Conference Proceedings

TeX Users Group

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

Memberships and Subscriptions

2006 dues for individual members are as follows:

- Ordinary members: $85.
- Students/Seniors: $45.

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

Membership in the TeX Users Group is for the calendar year, and includes all issues of *TUGboat* for the year in which membership begins or is renewed, as well as software distributions and other benefits. Individual membership is open only to named individuals, and carries with it such rights and responsibilities as voting in TUG elections. For membership information, visit the TUG web site.

Also, (non-voting) *TUGboat* subscriptions are available to organizations and others wishing to receive *TUGboat* in a name other than that of an individual. The subscription rate is $95 per year, including air mail delivery.

Institutional Membership

Institutional membership is a means of showing continuing interest in and support for both TeX and the TeX Users Group, as well as providing a discounted group rate and other benefits. For further information, contact the TUG office or see our web site.

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
+1 206 203-3960

Electronic Mail
(Internet)

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

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

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

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

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

Have a suggestion? Problems not resolved?
The TUG Board wants to hear from you:
Please email `board@tug.org`.

[printing date: October 2006]

EuroTEX 2006 Proceedings

Debrecen, Hungary

July 5–8, 2006

TUGBOAT

COMMUNICATIONS OF THE TEX USERS GROUP

TUGBOAT EDITOR BARBARA BEETON

PROCEEDINGS EDITOR KARL BERRY

VOLUME 27, NUMBER 1 • 2006

PORTLAND • OREGON • U.S.A.

EuroTEX 2006: A Hungarian TEX Rhapsody

The XVIth European TEX Conference

Sponsors

DANTE e.V. ▪ GUTenberg ▪ MATEX ▪ TEX Users Group ▪
Faculty of Computer Science, University of Debrecen ▪
Hungarian Mathematics Society of Transylvania ▪
Panem Kiadó ▪ Gyöngyi Bujdosó

Programme Committee

Karl Berry ▪ Thierry Bouche ▪ Gyöngyi Bujdosó ▪ Luzia Dietsche ▪ Yannis Haralambous ▪ Hans Hagen ▪
Taco Hoekwater ▪ Bogusław Jackowski ▪ Jerzy Ludwichowski ▪ Péter Maczó ▪
Frank Mittelbach ▪ Bence Nagy ▪ Bernd Raichle ▪ Péter Szabó ▪ Ferenc Wettl

Organising Committee

László Baranyai ▪ Gábor Bella ▪ Gyöngyi Bujdosó ▪ Luzia Dietsche ▪ Klaus Höppner ▪
Péter Hanák ▪ Tamás Mihálydeák ▪ Bence Nagy ▪ Volker RW Schaa ▪ Péter Szabó ▪
Maria Jolanta Szelatyńska ▪ Ferenc Wettl

Preprints Committee

István Baranyai ▪ Gyöngyi Bujdosó ▪ Péter Szabó ▪ Ferenc Wettl

Participants

Gábor Bella, ENST Bretagne
Thierry Bouche, Université de Grenoble
Gyöngyi Bujdosó, University of Debrecen & MATEX
Hossam Fahmy, Cairo University
Katalin Fried, Eötvös Loránd University
Hans Hagen, Pragma ADE & NTG
Yannis Haralambous, ENST Bretagne
Hartmut Henkel, von Hoerner & Sulger GmbH
Maurizio Himmelmann, GUiT
Taco Hoekwater, Elvenkind BV
Jean-Michel Hufflen, University of Franche-Comté
Bogusław Jackowski, GUST
Péter Jeszenszky, University of Debrecen
Lajos Kollár, University of Debrecen
Siep Kroonenberg, Rijksuniversiteit Groningen
Dag Langmyhr, University of Oslo
Jerzy Ludwichowski, Nicolaus Copernicus University & GUST
Tamás Mihálydeák, University of Debrecen, MATEX
Ildikó Miklós, KöMaL
László Németh, OpenOffice.org team
Karel Píška, Czech Academy of Sciences
István Radó, Horváth & Partners
Arthur Reutenauer, École Normale Supérieure
Volker RW Schaa, DANTE e.V.
Martin Schröder, pdfTEX team
Péter Szabó, Budapest University of Technology and Economics
Ulrik Vieth, Germany
András Virágvölgyi, sequens.hu
Staszek Wawrykiewicz, GUST
Ferenc Wettl, Budapest University of Technology and Economics & MATEX

The Hungarian TEX Rhapsody — EuroTEX 2006

EuroTEX 2006 was the first international TEX conference held in Hungary; it was organized by the Hungarian TEX Users Group, MATEX. It took place during the first week of the hot Hungarian summer, from July 4 to July 8, in Debrecen, the second largest city in Hungary.

Debrecen is a town of several universities, and the capital of Calvinism. The conference was held in the new campus of the University of Debrecen, near the center of the town. During the excursion, the participants visited some of the nice places of the town: the old campus and buildings of the university, the old library and a school exhibition of the Theological University, and the main Calvinistic church. A short walk in the center was followed by a tour in the Hortobágy National Park. Here we could see the old Hortobágy bridge, as well as special, protected animals and a photo exhibition about the special flora and fauna of the area. And, of course, we got a taste of traditional Hungarian food.

The TEXers and typographers participating in the conference came from Europe and Africa: Czech Republic, Egypt, France, Germany, Hungary, Poland, The Netherlands, and Norway. We were very sorry that some of our friends could not participate because of personal reasons. We thank all the attendees for the talks, tutorials and valuable discussions that made the conference successful.

In this proceedings, papers on various parts of TEX and typography can be read. There are papers on how to use TEX for publishing journals and mathematical exercise databases, about attractive Roman and Arabic and even some bovine fonts, new developments in pdfLATEX, ConTEXt, Ω and MlBibTEX, and new hyphenation techniques for TEX and Ω. The other main thread of the conference was typography: papers about typographical history, aspects and systems can also be found in the proceedings. We regret to inform the reader that Yannis Haralambous's greeting lines delivered in Hungarian at the beginning of his presentation are not part of the proceedings! We hope that you will enjoy the papers as we enjoyed the excellent talks during the conference.

Beyond that many thanks must go to the members of the programme and organizing committees (listed on the preceding page) for their work in making the conference successful and valuable.

We are also grateful to the sponsors of the conference. Many thanks to the Faculty of Computer Science of University of Debrecen for ensuring the technical support and the place of the conference. We are thankful for the generous contributions from TEX user groups that made it possible to organize the conference in Hungary, namely DANTE e.V., GUTenberg, NTG, and TUG for financial aid, and TUG for publishing the proceedings. We also thank the Panem Publishing Company and the Hungarian Mathematics Society of Transylvania (EMT) for the financial support.

Special thanks to Yannis Haralambous, Karl Berry, Jerzy Ludwichowski and Maria Jolanta Szelatyńska for their help given to us before and during the organizing process. Many thanks to Hans Hagen, Klaus Höppner, and Volker RW Schaa for their help in ensuring the financial background, and Karl Berry and Barbara Beeton for their work in editing, correcting and publishing this proceedings. And we are grateful for the contributions of Luzia Dietsche, Taco Hoekwater, Bogusław Jackowski, Péter Maczó, Frank Mittelbach and Bernd Raichle.

We were honored by the participation of the esteemed typographer István Baranyai, who created the conference logo, buttons, and the design used for the preprints and these proceedings. Thank you, István!

And last but not least I must thank the work and help of my Hungarian TEX mates, to László Baranyai for ensuring the technical background, Gábor Bella for preparing and keeping the web pages up to date, Tamás Mihálydeák for many things in the local organizing work, Bence Nagy for the web page design, Péter Szabó for making the style file and editing the preprints, as well as setting up the mailing lists and other infrastructure, and Ferenc Wettl for his work on the committees and the valuable conversations.

Hope to see you at the next TEX conference!

— With best regards, Gyöngyi Bujdosó

Impressions from EuroTEX 2006

GÁBOR BELLA

Arrival

Having been responsible for programming the conference website and followed email discussions on organisational matters, I arrived at the conference with the knowledge that only around 30 people were registered, a fact that worried me slightly. It had also appeared that a single person bore the brunt of the organisational work. However, what cheered me up was that this person was none other than Gyöngyi, the leading lady of the Hungarian TEX community whose wonderful personality we all know. And indeed, as it turned out on arrival, everything was impeccably organised and, despite the small number of people present, the overall ambience was very positive, light, and relaxed. The conference was held on one of the campuses of the Debrecen University. Hotel rooms were also provided on campus, just 50 metres from the conference hall. I, for one, was perfectly satisfied with the facilities and living conditions.

Debrecen is one of the most important cities of Hungary, both in terms of history and present influence. It is situated some 250 km to the east of Budapest. Those who traveled to Debrecen by train had no big problems during their trip. However, attendees taking the Budapest-Debrecen flight had the unique experience of landing *twice*. The Debrecen airport is new (it opened only two months before the conference) and apparently the logistics arrangements are not yet perfect: the flight controllers simply forgot to give landing clearance to the aeroplane, forcing it to head upwards again, do a circle and proceed to land a second time.

The first afternoon's tutorial sessions were followed by a dinner and a collective watching of the Germany vs. Italy semi-final. The author has to doff his hat to all the German supporters at EuroTEX who soberly accepted (despite the quantity of cheap Hungarian beer consumed) the unfavourable results of the game without the least inclination towards football hooliganism.

Registration

The registration procedure of TEX conferences is always a little bit like Christmas: this is the time when participants get to see and collect the usual T-shirts, mugs, pens, and other lion-related memorabilia. This year, these items all had a genuine Hungarian touch to them: instead of usual conference handbags, people were given haversacks. T-shirts were not bearing a traditional Duane Bibby illustration; rather, the conference logo was a simple design referring to the lions of the Chain Bridge, the symbols of Budapest and of Hungary (this is also proof of the strong TEX community that has existed here since the end of the 19th century). Conference mugs were handcrafted locally. Even the proceedings volume will perhaps become a collectible item, not only because of the simple but elegant design (adapted also for this proceedings) but also because it opens with a mystery: if this was really the 16th EuroTEX conference, why was a number *XVII* printed on the title page? Does EuroTEX feel too young and thus cheat to appear older? (The person who finds the most satisfying answer to this puzzle will win an extra set of conference items.)

Talks

Usually, talks at TEX conferences are mostly technical but sometimes also have a lighter side. In Debrecen, we had our share of both serious and fun talks. It also seemed that participants in general shared a genuinely positive and optimistic view about the present and future of TEX-related software development. As you will see from the following articles, we heard several announcements and saw demos of very exciting new advances. Fonts and font-related development were perhaps the most popular subject, addressed by seven speakers. Other hot topics were better extensibility of TEX-related systems and improved hyphenation algorithms, showing the need to address current weak points of TEX-based systems. Quite a few of the talks dealt not with the development of TEX itself but with using it for a variety of practical purposes. Read and enjoy.

Nightly sessions

When there was no football game to watch in the evening, we headed out from the campus into the Debrecen nightlife. The session chair for these nightly events was Gyöngyi, who led us around the city and took us to various pubs and bars. Not everyone, however, was equally motivated to stay up late, and only the most adventurous got to taste *Unicum*, a strong Hungarian herbal spirit whose medicinal powers were perhaps less immediate than its side effects. Congratulations to Arthur, the most fearless drinker of the group!

Trip

An entire day was devoted to exploring the city of Debrecen and its surroundings. First we had a look at the university's impressive central building. Having its roots in the historic *Reformist College* founded almost 500 years ago, the University of Debrecen is one of the biggest in Hungary, with more than twenty-five thousand students. Its enormous library (the second largest in the country) with its rich catalogue of ancient and rare books, including a large collection of prints from various European and Hungarian masters of typography, was a special treat for us.

In the afternoon we headed out to a National Park that embraces the westernmost remnant of the Eurasian *steppe* (*puszta*). One of the park's main goals is to preserve the special cattle breeds that do not exist elsewhere. (Cows were another recurrent theme at the conference; see both Taco's talk and László Németh's presentation featuring an *ice cream cow* that we now all know how to hyphenate in Swedish.) The park also gives insights into how herdsmen and locals lived there centuries ago. And, *of course*, we all had *gulyás* for lunch!

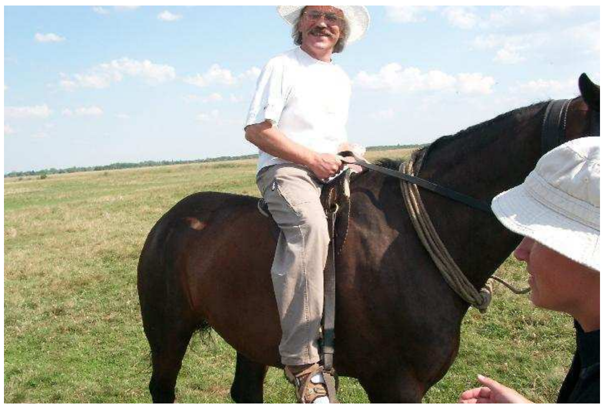
Banquet dinner and conference ending

The banquet dinner provided, in the author's opinion, a real high point to the conference: the very nice dinner (especially the wild boar chops) was crowned by a real surprise when, with a glass of red wine in hand, we headed to a nearby hall to attend an exclusive live concert by a renowned local Dixieland band. It was a treat to listen to such professional musicians; those who did not come to the meeting missed a unique experience.

The next day, though full of interesting talks, was the day of departure. Despite the small number of attendees, I believe it was a very successful conference. I left with very good impressions and I believe all of us felt the same. *Thank you, Gyöngyi, for all your hard work, and see you all next time!*

(A sampling of conference photos are below, courtesy of Hartmut Henkel, Volker Schaa, Martin Schröder, and Ulrik Vieth. Many more are at `http://www.matexhu.org/eurotex2006/pictures`. *Ed.*)

Julika, Zsuzsa and Beáta at the registration desk


Staszek Wawrykiewicz

At work or watching football?

Martin Schröder and Hartmut Henkel

A technical problem needs the experts: Tamás Mihálydeák, Jerzy Ludwichowski, Hans Hagen, Yannis Haralambous

Tales from the history of typography: András Virágvölgyi

Thierry showed us the perspectives: Jean-Michel Hufflen, Thierry Bouche, Ferenc Wettl, Hossam Fahmy, Staszek Wawrykiewicz, Gábor Bella

Dag Langmyhr, Bogusław Jackowski, Jerzy Ludwichowski

The four champions at cracking the whip:
Jerzy Ludwichowski, Arthur Reutenauer, Hossam Fahmy, Ferenc Wettl

Dixieland in Debrecen

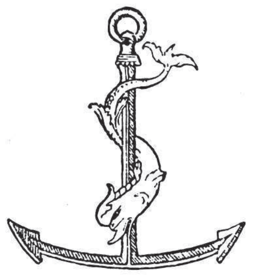
Arthur Reutenauer, Ulrik Vieth, Taco Hoekwater, Staszek Wawrykiewicz, Volker Schaa, Gyöngyi Bujdosó, Gábor Bella

At the center, before departure: Hartmut Henkel, Arthur Reutenauer, Siep Kroonenberg, Hans Hagen, Taco Hoekwater, Hossam Fahmy

Gyöngyi Bujdosó

Would Aldus Manutius have used TeX?

ANDRÁS VIRÁGVÖLGYI
historian, designer
www.sequens.hu



Aldus' emblem, the dolphin curling around an anchor

ABSTRACT
Traditional Typography · Golden Mean · Linear Reading · The Siege of Constantinople · Aldus Manutius · His Ambitious Publishing Program · The Most Beautiful Book of the World · Festina Lente · Would Aldus have used TeX?

It might be of interest to the participants attending the 16th EuroTeX conference to hear about the golden age of traditional typography in the 16th century and get to know one of the most famous book publishers of all times, Aldus Manutius of Venice.

The word typography is a compound of Greek elements. *Typos* means engraved illustration and *grapho* means to write.[1] *Scrivere sine penne* (to write without a pen), as they called it in the 16th century. Typography uses predefined shapes, letters, texts and sometimes illustrations as well. As final output it creates the printed page, which represents another level of visual quality. After the new types of books of the 16th century left behind the poor readability and ponderousness of old codices, typographers, who at the time were the printers themselves, followed the rules of the retrospectively labeled "traditional typography". It was a combined outcome of Renaissance modern style and the lasting effect of medieval lettering. "Traditional typography" determined the encounters of many generations with letters and enjoyed a consciously or unconsciously accepted status in the eyes of readers throughout the centuries.[2]

According to the traditional style the design of books starts with choosing the right proportions for page margins. Setting the inside, outside, top and bottom margins will give us the type area. The height, width and carefully chosen position of the type area influences overall proportions and balance of the book. Not everyone and not every workshop took the time and the trouble to carry out this kind of precision work and fine tuning. As time went by well-tested recipes started to form, incorporating certain elements of antique architecture and epigraphy. The *golden mean* for example, very popular among Renaissance artists also found its way to book design. It stated that the smaller portion should compare to the bigger as the bigger compares to the whole, as in the series 3 : 5 : 8. The relevance of these proportions could be observed in nature or in antique architecture. Like sculptors, painters or architects before them, typographers also wanted to take advantage of this noble rule when designing letters, margins or title pages.

After they made their decision about margin sizes, they could begin to compose the text. Information was organized according to a tight hierarchy, starting with the biggest type size used on title pages through the opening pages of chapters and the type of the body text to the smaller fonts of marginal notes and indices. The rhythm was provided by the repeating order of chapter headings, subheads and paragraphs, all indicated by traditional typographic methods.

Printers
(16th century woodcut)

Further decisive features of the traditional page spread were symmetry and static arrangement. This structure encouraged linear reading, so readers of the 16th century did not feel at all that they missed something. They read a book thoroughly from cover to cover. The list of recommended books was short and they had the time to periodically re-read some of them.

[1] *Péter Virágvölgyi: The Art of Typography. [Osiris Handbooks] Budapest, 2002, Osiris Kiadó.*
[2] *Suzanne West: Working with Style. Traditional and Modern Approaches of Page Design. Budapest, 1998, UR Könyvkiadó. p. 54.*

Not to mention the Bible itself, which was ſtudied and reſtudied by many people every year.

Renaissance typography gradually filled the ſpace created by the newly invented art of printing. New and subſtantial methods of arranging information profoundly affected reception and the way of thinking of 16th century people juſt like the digital world influences our approaches nowadays.[3]

ALDUS MANUTIUS

In 1453 the Turks occupied Conſtantinople. Around the same time Gutenberg was preparing his 42-line Bible for printing. The taking of Conſtantinople not only coincided with the beginning of European printing, but it also had an indirect effect on it as well. During Byzantine times many Greek schools proſpered on the coaſt of the Boſphorus ſtrait. When the Turkish siege ſtarted, these Greek scholars left their schools and fled to Italy. They had an extensive knowledge of classical authors of ancient times. Therefore their relocation turned Venice into a true center of classical erudition and research. There was something in the air in that city and the situation juſt needed an entrepreneur to take advantage of it.

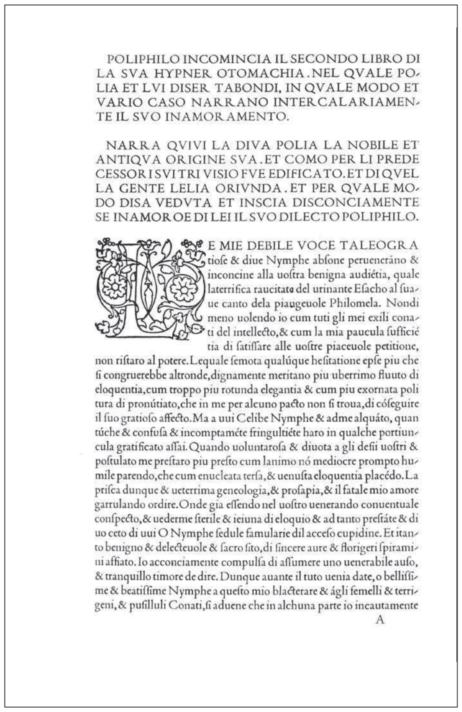
Aldus Manutius was a modeſt language inſtructor at the time, at the side of famous humaniſts like Pico della Mirandola. He taught them Greek and Latin and surely felt the need to create better editions to be used in education or for other humaniſts working with the same texts. Underſtanding the significance of all those Greek scholars arriving in Venice, he moved to the city to try his luck. At firſt he was employed by a Venetian book merchant and printer called Asola. He was later named the leader of Asola's workshop, and possibly because of this the owner's daughter became Aldus' wife. This was not an unusual ſtep though, as it was Asola's beſt way to ensure that his talented employee would carry on with his thriving workshop. Aldus, on the other hand, had navigated himself to a position from which he could set about executing his monumental plans. To secure a serious intellectual background he contacted Latin ſpeaking humaniſts as well as the aforementioned Greek emigrants. Now he could frequently consult Greek scholars of Conſtantinople and also hired scribes from the island of Crete as Greek typesetters and proofreaders.

Aldus Manutius

The realization of his publishing program ſtarted in 1495. The result was the creation of probably the moſt beautiful and effectual books in the hiſtory of printing. Greek authors were published firſt. The six-volume Ariſtotle should be mentioned above all, which used a Greek cursive type modeling those scribes' original handwriting, and appeared in print between 1495 and 1498, causing a big sensation among European humaniſts. This initial success was soon followed by editions of Sophocles, Plato and Thucydides. After them came other classical authors writing in Latin: Virgil, Horace and Ovid. The size of each run was around a thousand copies. A surprisingly modern publishing policy governed these early critical editions. Aldus, being a major representative of the humaniſt approach to classical literature, preferred to get rid of all the medieval commentaries and foſter the reading of original texts in their original languages. Thus it was up to the reader to make up his or her own variant, and to interact freely with ancient authors. To help the learning of classical languages – since originally he was a teacher – he also published quality textbooks and dictionaries.

The moſt beautiful book of the world, as book hiſtorians like to call it, came out in 1499, four years after the initiation of Aldus' ambitious program. It contained Francesco Colonna's allegorical and mythic poem, the *Hypnerotomachia Poliphili.* Using 170 excellent renaissance wood-cut illuſtrations, fully harmonizing type design and margin pro-

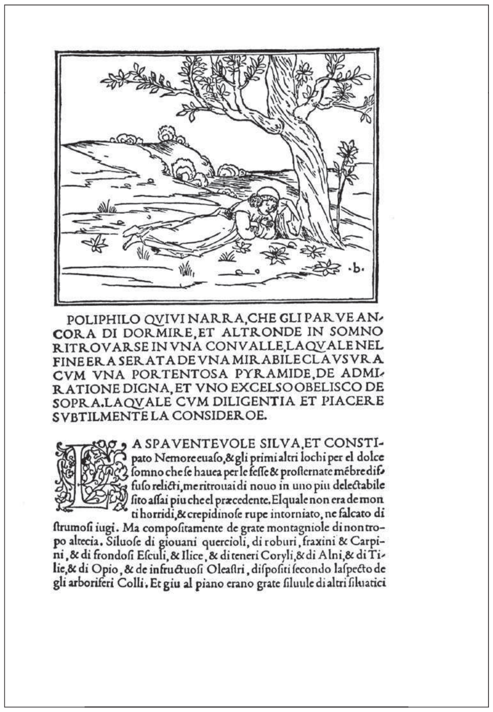
[3] *Leah S. Marcus: The Silence of the Archive and the Noise of Cyberspace. In: The Renaissance Computer. Knowledge and technology in the first age of print. Eds. Neil Rhodes and Jonathan Sawday. London, 2000, Routledge. p. 22.*


POLIPHILO INCOMINCIA IL SECONDO LIBRO DI LA SVA HYPNER OTOMACHIA. NEL QVALE POLIA ET LVI DISER TABONDI, IN QVALE MODO ET VARIO CASO NARRANO INTERCALARIAMENTE IL SVO INAMORAMENTO.

NARRA QVIVI LA DIVA POLIA LA NOBILE ET ANTIQVA ORIGINE SVA. ET COMO PER LI PREDECESSORI SVI TRIVISIO FVE EDIFICATO. ET DI QVELLA GENTE LELIA ORIVNDA. ET PER QVALE MODO DISA VEDVTA ET INSCIA DISCONCIAMENTE SE INAMOROE DI LEI IL SVO DILECTO POLIPHILO.

E MIE DEBILE VOCE TALEOGRAtioſe & diue Nymphe abſone peruenerãno & inconcine alla uoſtra benigna audiétia, quale laterrifica raucitate del urinante Eſacho al ſuaue canto dela piangeuole Philomela. Nondimeno uolendo io cum tuti gli mei exili conati del intellecto, & cum la mia paucula ſufficiétia di ſatiſſare alle uoſtre piaceuole petitione, non riſtaro al potere. Lequale ſemota qualúque heſitatione epſe piu che ſi congruerebbe altronde, dignamente meritano piu uberrimo fluuto di eloquentia, cum troppo piu rotunda elegantia & cum piu exornata politura di pronútiato, che in me per alcuno pacto non ſi troua, di cóſeguire il ſuo gratioſo affecto. Ma a uui Celibe Nymphe & adme alquáto, quantúche & confuſa & incomptaméte fringultiéte haro in qualche portiuncula gratificato aſſai. Quando uoluntaroſa & diuota a gli deſii uoſtri & poſtulato me preſtaro piu preſto cum lanimo nó mediocre prompto humile parendo, che cum enucleata terſa, & uenuſta eloquentia placédo. La priſca dunque & ueterrima geneologia, & proſapia, & il fatale mio amore garrulando ordire. Onde gia eſſendo nel uoſtro uenerando conuentuale conſpecto, & uederme ſterile & ieiuna di eloquio & ad tanto preſtáte & diuo ceto di uui O Nymphe ſedule famularie dil acceſo cupidine. Et itanto benigno & delecteuole & ſacro ſito, di ſincere aure & florigeri ſpiramini affiato. Io acconciamente compulſa di aſſumere uno uenerabile auſo, & tranquillo timore de dire. Dunque auante il tuto uenia date, o belliſſime & beatiſſime Nymphe a queſto mio blacterare & ágli femelli & terrigeni, & puſilluli Conati, ſi aduene che in alchuna parte io incautamente

A


POLIPHILO QVIVI NARRA, CHE GLI PARVE ANCORA DI DORMIRE, ET ALTRONDE IN SOMNO RITROVARSE IN VNA CONVALLE, LAQVALE NEL FINE ERA SERATA DE VNA MIRABILE CLAVSVRA CVM VNA PORTENTOSA PYRAMIDE, DE ADMIRATIONE DIGNA, ET VNO EXCELSO OBELISCO DE SOPRA. LAQVALE CVM DILIGENTIA ET PIACERE SVBTILMENTE LA CONSIDEROE.

A SPAVENTEVOLE SILVA, ET CONSTIpato Nemore euaſo, & gli primi altri lochi per el dolce ſomno che ſe hauea per le feſſe & proſternate mébre diffuſo relicti, me ritrouai di nouo in uno piu delectabile ſito aſſai piu che el præcedente. Elquale non era de monti horridi, & crepidinoſe rupe intorniato, ne falcato di ſtrumoſi iugi. Ma compoſitamente de grate montagniole di non tropo altecia. Siluoſe di giouani quercioli, di roburi, fraxini & Carpini, & di frondoſi Eſculi, & Ilice, & di teneri Coryli, & di Alni, & di Tilie, & di Opio, & de infructuoſi Oleaſtri, diſpoſiti ſecondo laſpecto de gli arboriferi Colli. Et giu al piano erano grate ſiluule di altri ſiluatici

Pages from the Hypnerotomachia Poliphili, one of the most beautiful book of the world

portions, it received a form that still evokes the admiration of today's typographers and bibliophile book collectors all over the world. Surprisingly enough, though the unique nature of *Hypnerotomachia Poliphili* was certainly noticed by contemporaries (a pirate edition came out soon after in Lyon), it was never reprinted in Venice during the next hundred years.

Aldus' achievements deserve acknowledgement already, but his most remarkable innovation is still to follow. Production of books in the 16th century ever becoming cheaper and faster, workshops could start counting on bigger audiences. He was first to realize that instead of the large-format books printed before, readers who preferred solitary reading needed portable or pocket-size editions – as we call them today. Fifty years passed after the appearance of Gutenberg's enormous, two-column 42-line Bible, when the Aldus Officina in Venice, leaving the old codex format behind, started to produce the incredibly popular one-column pocket-size editions of the classics.

Soon the *Aldi Neacademia*, a distinguished group of scholars (men of letters) was formed. By this time the workshop's Greek and Latin consultants had daily meetings to decide about the titles to be published. While the medieval scholar accumulated, the Renaissance humanist judged the old manuscripts: they reconsidered the classic authors known in medieval times and tried to acquire previously unknown works by research or purchase. This was the way *editio princeps* books (first printed versions of the classics) got published, considered rare gems by later collectors. However, they did not refuse to publish eminent medieval authors either; we can find Dante and Petrarch in Aldine editions. The farseeing editorial policy had its fruitful result, the carefully selected titles were appreciated all over Europe.[4]

Characteristic of the success of the pocket-size editions is the large number of imitators especially in France. Simone de Colines launched a similar series in Paris, while Aldus had to defend himself against the pirate editions of the Lyon workshops by issuing a public protest letter. In this he enumerated the errors made by the printers of Lyon, so that an original copy could be easily discerned from a fake one. The antecedent of the pirate editions, of course, was the appearance and success of Aldine publications in the French market. Classical Roman culture crossed the Alps in the form of these beautiful books, and as Beatus Rhenanus, the biographer of Erasmus, put it: "northern barbarians" could now learn Latin and Greek and save a long journey to Italy.

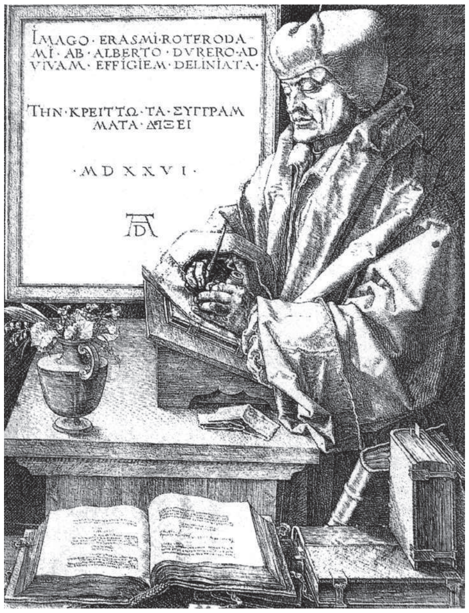
Being an Italian, Aldus was a faithful supporter of antiqua letters. In the beginning he used fonts strikingly similar

[4] *Martin Lowry: The World of Aldus Manutius. Oxford, 1979, Oxford University Press.*

to those used by the French Nicholas Jenson, also working in Venice. In 1501 he commissioned the talented letter designer, Francesco Griffo (1450–1518) to create a new Latin typeface. This was the first antiqua typeface which used capitals somewhat smaller than the ascenders of lower case letters. In the next year Griffo designed the famous cursive or *italic* type as it was later called, honoring the country of origin. Based on the so-called *cancelleresca*, it became the printed version of humanistic handwriting. Today we use italics to put emphasis on words or passages of text, but Aldus actually used it for typesetting whole books. Italics started its career in his workshop, exercising a great deal of influence on 16th century typography and fostering the victory of antiqua type all across Europe.

Page numbering also spread by Aldus' books – he was among the first to recognize its practical importance. Page numbers, apart from assisting the work of bookbinders, made it much easier for readers to refer to a given section within a book.

The Aldus workshop had its heyday at the turn of the 15th and 16th centuries. In the Juvenal edition of 1501, Aldus set forth his philosophy and goals, many of which were already achieved by releasing the series of elegantly designed and carefully edited classic publications. His logo, the dolphin with the anchor, appeared in his books from 1502. In the eyes of book lovers this mark represents utmost excellence in terms of both content and form. The dolphin curls around the anchor and emphasizes the classic saying which goes with it: *Festina lente* [*gr. Speude bardeos*], i.e. to hurry slowly. It was Erasmus[5] who made this saying popular. Originally it comes from a play called *The Knights* by Aristophanes: "Speude takheos" – to hurry up quickly. In its reversed form the saying has several meanings. According to Erasmus this stoic statement should prevent princes from acting in the heat of the moment, to avoid swift and arbitrary decisions. Fabius Maximus is mentioned as the best example for slow diligence, who continuously weakened the invading army led by Hannibal employing his distressing technique. That is why he was labeled the postponer *(cunctator)* by Roman politicians. Supposedly emperor Augustus


IMAGO · ERASMI · ROTERODA
MI · AB · ALBERTO · DVRERO · AD
VIVAM · EFFIGIEM · DELINIATA ·
ΤΗΝ · ΚΡΕΙΤΤΩ · ΤΑ · ΣΥΓΓΡΑΜ
ΜΑΤΑ · ΔΕΙΞΕΙ
· MDXXVI ·
AD
Portrait of Erasmus of Rotterdam by Albrecht Dürer (1526)

and Vespasian also liked the saying *Festina lente.* The dolphin curling around the anchor appears on one of the coins issued by Vespasian.

Posterity highly esteems Aldus' publishing activity. Jacob Burckhard (1818–1897), one of the earliest and most famous researchers of the Renaissance period, highlighted his importance. Several contemporary memorials record the exceptional popularity of Aldine books. The following quotation comes from a letter written by the German humanist Heinrich Glareanus to Ulrich Zwingly, dated on 19th October, 1516: "I cannot miss to mention, that Wolfgang Lachner, our Frobenus' father-in-law ordered a wagonful of classics from Venice, the best of Aldus' publications. If you would like to have some of them, let me know quickly and send cash. Because as soon as a similar shipment arrives, there are already thirty people surrounding it and keep asking 'how much is it?' then start to fight over it. Passion rapidly ignites animated discussions and often seizes men who cannot even understand them."[6]

As an author and consultant Erasmus worked together with the Venetian printer and publisher several times. He already wrote to Aldus, that his translations of Euripides would make him immortal, especially if they were "printed

[5] *Stefan Zweig: The Glory and Tragedy of Erasmus of Rotterdam. Budapest, 1993. Holnap Kiadó. pp. 73–80.*

[6] *Nándor Várkonyi: The History of Books and Letters. Budapest, 2001, Széphalom Könyvműhely. p. 352.*

using your small [minutius] types, which are the most elegant in the world *(tuis excusae formulis… maxima minutioribus illis omnium nitidissimis)*." But beauty is not everything. Erasmus calls the octavo editions the outstanding products of his age. Not even Ptolemaios of Philadelphia could access literature and science the way Aldus Manutius made it possible in the early modern age. While the great king had built only one extensive library, Aldus raised 'a library without walls,' which will survive all disasters. Willibald Pirckheimer, yet another humanist, esteemed his Theokristos-edition so highly, that he asked the famous German engraver and painter Albrecht Dürer to illustrate its cover and design an *ex libris* sign for it. Every detail of the bucolic idyll created by Dürer follows the text faithfully.

There was feedback from Hungary, too. Sigismund Thurzo, provost and secretary of the king, writes in his grateful letter of 1501: "My different kinds of affairs consume the time I could spend at home in the company of poets and orators. Your books – being very practical, so that I can take them with me for my walks or have them around during conversations or my affairs in the court – cause me much delight." [7]

No doubt, Aldus Manutius was the leading publisher in Europe at the turn of the 15th and 16th centuries. A generation after Gutenberg, printers overstepped the traditional manuscripts of medieval times and by appeasing the needs of this new generation of readers, published Latin and Greek classics with flawless content and in wonderful form.

Aldus died in 1515, but his workshop continued to operate throughout the 16th century. His work, if not to the same effect but still at a very good standard, was carried on by his son and later by his grandchild for a hundred years. In 1597 the grandchild gave up the workshop in Venice and in response to the call of the pope he went to Rome to manage the printing facilities of the Vatican. By that time the humanistic movement was a thing of the past. Public opinion was mainly concerned with the struggle of the Catholic church with Protestants, and its efforts to renew itself.[8]

As they were popular, the number of surviving Aldine books is not too great and oftentimes they are worn-out because of frequent usage. But their quality is clearly shown by the fact that they were still good enough for the famous French playwright Racine (1639–1699), who got to know the Greek tragedies from Aldine editions nearly a hundred and fifty years after their publication.

Finally let us attempt to answer the question in the title of this paper. As we have seen, Aldus was seriously involved in the business of publishing 'scientific' books. He was open to the novelties of his trade; moreover he also made significant contributions to it with his inventions. Based on this we could assume that if computers, the main representatives of modernity and future, had existed at his time, he would have surely used them. But would he have used TEX?

Aldus published many works in Greek. (By the way, the word TEX comes from the Greek τεχνη or techné). Originally TEX, being an American software had a limit of working with 128 characters only. This could have raised initial problems for someone planning to bring out multi-lingual publications. However, this limit has been eliminated since then, and at this conference we hear about the both typographically and technologically difficult task of publishing the Koran itself.

Though I am not a TEX guru or a TEXpert, I understand the many advantages of TEX in the field of scientific publications, especially if they contain a lot of formulae. Well, we must realize that the early modern age of Aldus preceded Newton's scientific revolution. In the 16th century we could have possibly found some formulas in esoteric works by alchemists on how to make gold, but usually explained in very unclear terms. As far as the 'serious science' of the century is concerned, it was mainly represented by classical works of ancient authors. They did not perform experiments back then but read Aristotle instead, so science was more like literature.

We can state that Aldus' innovative personality was always ready to accept newer and better solutions. He would have thought about using TEX if Donald Knuth had been born several hundred years earlier. But since the works he published were not scientific but rather literary in form, he might have decided otherwise.

[7] *Anthony Grafton: Humanist Reading. In: Cultural History of Reading. Budapest, 2000, Balassi. Eds. Robert Chartier, Guglielmo Cavallo. p. 206.*

[8] *Lucien Febvre and Henri-Jean Martin: The Coming of the Book (L'Apparition du livre). The Impact of Printing 1450–1800. London, 1990, Verso. p. 124.*

The colourful side of TEX

KATALIN FRIED et al.
Faculty of Science, Institute of Mathematics
Eötvös Loránd University, Hungary
`kfried (at) cs dot elte dot hu`

The day it started…

First of all I would like to say thanks to all who helped me in my understanding and learning TEX. (My father, Ervin Fried, my husband, Lehel Juhász, friends, Gabriella Köves, Tamás Bori.)

It all happened in the late eighties. At that time Lehel worked as an editor at the publishing house of the Hungarian Academy of Sciences. One day he came home and with a smile on his face he said: I have something you are going to like. (You have to be aware that I started writing books right after I finished university — early eighties — and by this time I had been through two books.) This is, he said to me, a typesetting program. So what, I said, a typewriter can do it. But you can typeset mathematics symbols with it, he said. I can type mathematics on my typewriter, I said. But it looks nice!, he added. Now you are talking!, I said, but then I simply do not believe you.

Questions

Can you type a sum sign?, I asked. Yes, he said. Can you type integrals? Yes. Can you type matrices? Yes. Can you…, can you…, can you…, I kept asking. And the answer was always the same: yes, yes, yes.

After about a couple of hours I remembered something.

Just a few month before that I had problems about denoting arcs in a book. You know what I mean, taking an arc of a circle between the points A and B you would like to refer to this arc and denote it in a similar way as you denote a line segment, $\overline{AB}$, but with an arc above the letters. So I asked, can you typeset such an arc? No, he said. But I can typeset $\widehat{AB}$ or $\widetilde{AB}$. That is not good, I want to see an arc! Like $\overset{\frown}{AB}$. Just much nicer!

But how can you do such a thing?

And there is more! When simplifying fractions I need to cross out the numerator and the denominator. Then I have to write the new numerator and denominator above and below the fraction, respectively. You know, like

$$\frac{6}{8} = \frac{\overset{3}{\cancel{6}}}{\underset{4}{\cancel{8}}} = \frac{3}{4}.$$

But *how* can you do it?

And can you put a frame around a text? (You must not forget that at the time we only had plain TEX and no utilities.) You know, like $\boxed{A \neq B}$. Or rather $\boxed{\ A \neq B\ }$.

Yes, yes, but *how do you do it?*

Some answers

At the time when these questions arose we had no help at all. All we had was *The TEXbook* — and only I read English. For framing things we simply had to put them into a box then "wrap them into hrules and vrules".

```
\def\boxit#1#2#3\hfill\break
{\vbox{\hbox{\vrule\vbox{\hrule%
\kern#2\hbox{\kern#3#1\kern#3}
\kern#2\hrule}\vrule}}}
```

It did not take more than a couple of months to solve this problem. And refining took only another 2–3 years.

Now, crossing out the numerator and denominator of a fraction took somewhat more time. We had to wait until Eberhard Mattes created his version of TEX in 1990: emTEX.

Its `\special` feature gave us the freedom to create new graphic objects. With these we could solve some of our problems — similar to the framing problem.

We could define nodes:

```
\def\node#1{\special{em:point #1}}
```

We could draw lines between two nodes:

```
\def\line#1#2{\special{em:line #1,#2}}
```

Fractions could be "simplified graphically". We had only to measure the numerator and the denominator in TEX — by boxing it and then measuring the box itself — and then draw a line between the appropriate nodes.

And we could even import bitmap graphics into TEX. This was fairly important because we could not draw everything under emTEX.

And from this point on we could create drawings of polygons and lines. We could add letters and symbols to our drawings.

We only had to face one serious problem: how to position the nodes we want to use.

```
\def\put(#1,#2)#3{\vbox to0pt
{\vss\kern#2pt\kern#2pt\hbox
to0pt{\hss\kern#1pt\kern#1pt
\vbox{#3}\hss}\vss}}
```

did the trick for us. Notice that this is basically the same idea as the one we used in framing.

But we still could not draw arcs. So we still did not have the arc above *AB*.

It was about this time when we discovered a drawing utility for TEX: PICTEX.

It had wonderful features:

1) You could define a plot area to be used. The picture had height, depth, and width independent of our construction. Why is it important? Because an object that has no height, depth, and width is difficult to input into your TEX file. As soon as you have to position the picture and the text you are going to face problems.

If it has height, depth, and width you can handle it as a TEX object and so you can fit it into your text. True, it is still a hard job. (Of course it is easier if you just centerline the picture.)

2) You could draw circles and ellipses. What a joy! We could do elliptical arcs, circular arcs (that is, parts of ellipses and circles). Alas, still no arc above mathematical objects.

And what did we lose when switching to PICTEX? We had no nodes. That was a great loss so we started to use the two drawing programs at the same time.

PICTEX can be used under LATEX. But we got stuck in plain TEX. Forever, it appears ...

Demands of authors

Years have gone and we solved more and more problems concerning drawings. Seeing this, our authors have become greedier and greedier. Not only would they like us to create real drawings by computer (ones a graphic artist should do) but also, they would like to have colours added to their books.

Luckily, Lehel had his diploma in art, so he could create all kinds of drawings (only he didn't have time to draw, as he was busy "TEXing"). But we gave it a try. There were two things we tried:

1) Drawing, scanning, retouching and importing the drawing into TEX.

2) Drawing by a graphic program and importing the drawing into TEX.

(After these experiments we could import "anything" into TEX.)

We imported bitmap drawings as before. But the age of bitmap graphics was declining. We had to change to eps form. Luckily, we found Tomas Rokicki's epsf.sty file from 1989. (We started to use it many many years later — bitmaps did the trick for us for quite a while.)

The drawings were created in some graphic programs (like Illustrator, CorelDraw, etc.).

On the other hand, there was a need to do more mathematical objects, such as the graphs of functions and constructions.

We could not keep pace with the demands our authors set for us. emTEX and PICTEX were simply not enough.

But just around that time we found another software package perfect for our needs. This software was PSTricks by Timothy Van Zandt (from 1993). For using this we also needed a dvi → ps driver. Tomas Rokicki's "dvips" offered us the PostScript output.

What did we gain from it? Everything! It had all sorts of graphics abilities — all kinds of "PostTricks". We could embed PostScript code into the drawings! What joy we had!

A whole new world opened in front of our eyes with "posttricks" (pstricks).

More answers (fine tricks)

True, when doing constructions we had to face a new problem. Euclid's postulates give us the possibility

- to draw a line going through two given points: could be done.
- to open the compasses to a distance of two given points: not simple but could be done somehow.
- to draw a circle with a given diameter: not simple but could be done.

- to draw the intersection point (if any) of two lines: lacking!
- to draw the intersection points (if any) of two circles: lacking!

Our friend Tamás Bori gave us a hand. He created a utility with which we could construct the intersection points. We were drawing happily ever after...

No, we were not! We found that we could not draw in 3D. I am not a mathematician for nothing. I studied projective geometry. I know how to do the transformation on the 3D coordinates to create such a drawing. So we wrote and used the program. As curves were not given by their coordinates no curves could be drawn. (I have to mention that about a couple of years ago we found a 3D graphing program written by someone else.)

Programming TEX reminds me of a conversation I had with a colleague of mine: At the university everybody uses TEX and when I told him I write programs in TEX with variables and calculations and such, he was astonished. 'Can you really write a program in TEX?' Well, not all of us do it. But for creating an animation I have to have a variable to calculate the number of phases, to calculate the measurement of objects changing, to calculate the shades of colours to use. Because that's what animation is.

And we could draw functions dot-by-dot. And we could create animation.

And what is a presentation? Properly animated pages. So, we can create a presentation. As a matter of fact, we have done such a presentation — like the one at this conference.

Open questions

I want to draw your attention to an important point: these programs have been available from the mid 90's. I have not seen anybody else using them. I believe that we are offered too much and we can take too little. Each of us finds small bits of all the knowledge that had been created in connection with TEX. We, ourselves have created utilities, tools for TEX we never published. Imagine what a huge amount of knowledge there must be!

Still one question remains: *how can you put an arc above AB?!?*

Finally, I would like to say thanks to all those who posed questions to me to make me think about TEX problems (and solve them, most of the cases).

Opening up the type[*]

TACO HOEKWATER
Elvenkind, Dordrecht
`taco (at) elvenkind dot com`

Abstract

In the near future, pdfTEX will gain the ability to use OpenType fonts. This paper explains in broad terms what will be done and why it will be done in that way.

Introduction

There are many reasons why OpenType support is a good thing for pdfTEX to have. For one, many of the latest commercial fonts can only be purchased in OpenType format. For another, most of the new OpenType fonts are of higher quality than their older brethren. Also — this is a minor point, but not completely unimportant — everybody else does it. So long as pdfTEX does not support OpenType, it has no chance of being perceived as a viable competitor to those other systems.

Finally and perhaps most importantly, OpenType allows us to escape from TFM format and its many limitations, without the need to invent another special font format that can only understood by TEX and not much else.

The plan is for the OpenType support to be included as part of a project to create a high-end Arabic typesetting engine based on a merge of pdfTEX, Aleph and luaTEX. The final result of this project will be open source and can be merged into future versions of pdfTEX.

Wishes and constraints

Of course we want to integrate the new OpenType support such that it behaves well with the rest of TEX:

- It should be possible to use all of the glyphs and features in the font.
- The implementation, its input, and its output should all be platform independent: same font, same syntax, same typesetting.
- We want to make the implementation extensible, just in case someone comes up with OpenType^{++} in the next decade.
- It should still be possible to define virtual fonts and use the current pdfTEX micro-typography features like protruding and font expansion.
- The ability to make run-time adjustments to the font characteristics is desirable.
- The interface should be usable by somebody who is not trained as a programmer.
- For nice and small pdf output, some sort of font subsetting has to take place.
- Backward compatibility code for traditional TEX and PostScript fonts has to remain; the goal is evolution, not revolution.
- We wish full control at the basic glyph level, not limited to turning OpenType features on or off.

[*]This work is made possible by a grant from Colorado State University.

Unicode

When dealing with OpenType fonts, adding support for Unicode is more or less implied. Therefore pdfTEX had to be extended to handle Unicode input as well as output.

File I/O

Partial Unicode support is already in the current luaTEX code base:

- UTF-8 encoded text input and output.
- Characters and tokens use 21 bits for storing character information.
- Hyphenation patterns are loaded in UTF-8.
- The pool file (strings) and buffer are Unicode enabled.

Characters

At present, TEX does not have a concept of a character: a 'charnode' is actually a glyph in a font, together with the font it should be taken from.

To fix this unpleasant intermingling of glyphs with characters, pdfTEX will be extended with two new node types:

- A font node is the result of a font selection command by the user.
- A unichar node is the result of tokens being read.

The 'charnode' functionality is still present (renamed to glyph node), but the new types will not be converted to the traditional TEX glyph, font pair until after the hyphenation pass is completed.

Hyphenation

The old node list and paragraph building routines intertwined ligature building, hyphenation and line breaking. On top of that, hyphenation patterns were stored using the 'charnode', as mentioned in the previous paragraph.

This resulted in a few unfortunate side-effects:

- patterns are font encoding dependent;
- hyphenation is impossible unless a `\hyphenchar` is present in the current font;
- hyphenation patterns can only use 256 characters at a time.

The new code separates hyphenation from the line breaking decisions: First it finds *all* potential hyphenation points in the words (made up of unichar nodes) and insert `\discretionary` nodes for all of them. Only after that step is completed will it attempt to find ligatures and break the paragraph into lines.

This change makes hyphenation completely independent of the current font.

A different internal representation of the loaded patterns will make it possible to use the full range of Unicode characters in hyphenation patterns as well as making it possible to extend the patterns in a language at run-time.

Languages

We believe this is a good opportunity to also tackle another traditional problem in TEX: the `\lccode`, `\uccode` and `\sfcode` tables. These tables contain information that is conceptually part of the current language, and should not be stored in a font attribute.

We want to increase the importance of `\language` codes and attach much more information to language switches. Other candidates for inclusion in language switching are the `\uchyph` parameter and the list of applicable ligatures.

Scripts

pdfTEX currently uses the TEX–XET algorithm from ε-TEX, with the primitives `\beginL` and `\beginR`.

This will be removed in favor of the much more advanced and flexible Aleph/Omega1 typesetting direction commands `\pagedir`, `\pardir`, etc.

An equivalent to ΩTP processing will be implemented using lua instead of ΩCP (precompiled binary) files.

Font loading

In current pdfTEX, fonts are internally represented as a large storage heap with a few dozen auxiliary tables that store various meta-information and pointers into the heap. All of those are global, and implemented as static objects.

While this is very efficient in terms of speed, it is also very hard to alter a font after it has been loaded, and the unification forces all fonts to offer strictly the same interface.

In the new setup, fonts will be loaded under the direct control of lua code, and they will be presented to the typesetting engine as a single lua table for each loaded font. This table will make the font behave much like an object that can be queried and altered directly by the macro programmer, either from TEX macro code (through `\fontdimen`) or from lua code (through callbacks from the typesetting engine).

The low-level font loading routines will be written in compiled C code, perhaps by using a separate library like freetype.

Conclusion

We gratefully acknowledge that this work is made possible by a grant from Colorado State University, with the sponsorship of Idris Hamid, and with support from TUG. A test version of the changes described in this paper should be available before the TUG 2006 meeting in Morocco. People wishing to stay up to date with respect to this project are invited to visit `http://www.luatex.org`.

MetaPost developments — autumn 2006

TACO HOEKWATER
Elvenkind, Dordrecht
`taco (at) elvenkind dot com`

Abstract

The new release of MetaPost includes some new features as well as a number of bug fixes. The new functionality includes: the possibility of using a template for the naming of output files; support for CMYK and greyscale color models; per-object PostScript specials; the option to generate Encapsulated PostScript files adhering to Adobe's Document Structuring Conventions; the ability to embed re-encoded and/or subsetted fonts; and support for the GNU implementation of troff (groff).

Introduction

Version 0.901 of MetaPost was released at BachoTEX 2005. It featured an updated manual and the new `mpversion` primitive, but was mostly a bug-fix release.

At that time, a new version was promised for the autumn. In hindsight, that was overly optimistic. It is now autumn 2006, and version 1.0 will finally be released by the time this article is published.

Bug fixes in version 1.0

Stability issues

In previous versions of MetaPost, the size of the memory array was not stored in the mem file, because it was assumed to be a fixed quantity. But in Web2c-based systems, the memory sizes are dynamic: the actual size that should be used by the executable can change depending on the command-line invocation and `texmf.cnf` settings.

This caused a number of bugs, for example

- disappearing specials from the output;
- incorrect error messages;
- unexplained crashes.

The required minimum memory size is now included in the memory dump file. If a mismatch occurs, an error message will be issued.

turningnumber

The previous (0.9) MetaPost executable used a very simple algorithm to implement the `turningnumber` operation: connect the dots, add up all the angles, and then divide by 360. This was a temporary patch that was added as a stop-gap measure to make the erroneous cases easier to predict. The current version includes a final fix for `turningnumber` by including new code that calculates the true curvature for path segments.

This new algorithm is based on a mailing list discussion between members of the group. It is slower, but (finally) 100% accurate.

Adobe Document Structuring Conventions

Thanks to detailed comments from Michail Vidiassov, the output will now strictly adhere to the Adobe Document Structuring Conventions for Encapsulated PostScript files — when the internal quantity `prologues` is set to 2 or higher. The special setting of `prologues` is needed for compatibility with existing MetaPost post-processing tools.

Ignored withcolor

All previous versions of MetaPost failed to recognize that the user supplied a color specification when the color consisted of three zero-valued parts, as in this example:

```
draw fullcircle withcolor black;
```

In this case, no PostScript color selection command was output at all. If any surrounding command specified a different drawing color, that color would be used instead of black.

Current color trashed after clip

Previously, a `clip` command would completely destroy the internal graphic state. As a side effect, it would force the default color of the following operation to be black even if the surrounding document specified a different text color. This is now fixed.

Table 1: Escape sequences for `filenametemplate`

%%	A percent sign
%j	The current jobname
%⟨0-9⟩c	The charcode value
%⟨0-9⟩y	The current year
%⟨0-9⟩m	The numeric month
%⟨0-9⟩d	The day of the month
%⟨0-9⟩H	The hour
%⟨0-9⟩M	The minute

New features in version 1.0

File-name templates

The first of the new features is support for output filename templates. These templates use `printf`-style escape sequences (listed above) and are re-evaluated before each `shipout`. Numeric fields may be left-padded with zeroes.

The new primitive is `filenametemplate`, and it is string-valued. Here's an example:

```
filenametemplate "%j-%3c.eps";
beginfig(1);
  draw p;
endfig;
```

If this input file is saved as `test.mp`, then the output file will be named `test-001.eps`, instead of `test.1` as in previous versions.

Here are the escape sequences:

To ensure compatibility with older files, the default value of `filenametemplate` is `%j.%c`. If it is assigned an empty string, it will revert to that default.

CMYK color model

Support is now provided for the industry-standard CMYK color model. Following is a simple example.

```
beginfig(1);
   draw fullcircle
      withcmykcolor (1,0,0,0);
endfig;
```

To make more flexible use possible, a new type of expression is introduced. A `cmykcolor` is a quartet of `numeric` values that behaves similarly to the already existing type `color`. This example is equivalent to the previous one:

```
beginfig(1);
   cmykcolor cyan;
   cyan := (1,0,0,0);
   draw fullcircle withcmykcolor cyan;
endfig;
```

The new `cyanpart`, `magentapart`, `yellowpart` and `blackpart` primitives allow access to the various pieces of a `cmykcolor` or the CMYK component of an image object.

Greyscale color model

Only two new primitives are needed for greyscale support: `withgreyscale` and `greypart`. This is because greyscale values are simple `numeric` values.

```
beginfig(1);
   faded := 0.5;
   draw fullcircle withgreyscale faded;
endfig;
```

Mark-only color model

There is also a new primitive for 'mark-only' support: `withoutcolor`. This command is convenient in cases where MetaPost is not supposed to output any explicit color commands to the PostScript file at all, as in the generation of font outlines. Example:

```
beginfig(1);
   draw fullcircle withoutcolor;
endfig;
```

Other color handling changes

A new primitive `defaultcolormodel` is introduced. This specifies the assumed color model for objects that are drawn without any color specification. In all three models that actually specify a color, the default color is black.

The primitives for the already existing RGB color model are now also available under new names: the draw option `withcolor` becomes `withrgbcolor`, and the variable type `rgbcolor` is an alias for `color`.

The existing primitive `withcolor` has been extended so that it accepts any of the five possible input syntaxes:

Actual input	*Remapped meaning*
withcolor ⟨rgbcolor⟩	withrgbcolor ⟨rgbcolor⟩
withcolor ⟨cmykcolor⟩	withcmykcolor ⟨cmykcolor⟩
withcolor ⟨numeric⟩	withgreyscale ⟨numeric⟩
withcolor ⟨false⟩	withoutcolor
withcolor ⟨true⟩	⟨nothing⟩

An image object can have only one color model. The last specification of `withcolor`, `withcmykcolor` or `withgreyscale` controls the color model for a particular object.

Object specials

The new MetaPost supports specials that can be attached to main drawing objects. These specials are output on their own lines, immediately before and after the object they are attached to.

The names of the two new drawing options are `withprescript` and `withpostscript`. Their arguments are simple strings that are output as-is. It is up to the macro writer to make sure that the generated PostScript code is correct.

```
beginfig(1);
  draw fullcircle
     withprescript "gsave"
       withpostscript "grestore";
endfig;
```

Multiple prescripts and postscripts for a single object are possible; simply repeat the command. They are placed in the output file in the same order in which they are specified.

Standalone EPS

If `prologues` is set to the value 2, MetaPost now generates a proper Encapsulated PostScript level 2 image that does not depend on `dvips` post-processing. A private PostScript dictionary will be created to reduce the output size for large images.

In this output mode, fonts are not actually embedded, but their definition will be handled correctly.

Font re-encoding

If `prologues` is set larger than `1`, any used fonts are automatically re-encoded, and the encoding vector file specified in the fontmap entry will be embedded in the output file.

This code is based on the font library used by dvips and pdfTEX. Also following in the footsteps of pdfTEX, there are two new associated primitives: `fontmapfile` and `fontmapline`. Their string-valued arguments use the same optional flag as pdfTEX:

	replace the current font list completely
+	extend the font list, but ignore duplicates
=	extend the font list, replacing duplicates
−	remove all matching fonts from the font list

Here is an example:

```
prologues := 2;
fontmapfile "+ec-public-lm.map";
beginfig(1);
  draw "Helló, világ" infont "ec-lmr10";
endfig;
```

Font inclusion

Font inclusion is triggered by `prologues` being equal to 3. Whether or not actual inclusion and/or subsetting takes place is controlled by the map files. These can be controlled using the syntax explained in the previous section.

GNU groff support

The new version of MetaPost has native support for GNU groff, thanks to a set of patches from Werner Lemberg and Michail Vidiassov.

Future plans

With version 1.0 out the door, plans for the next version are being made. The next release after this one is 1.1, and it will likely have the following set of new features:

MetaPost dynamic library

It will become possible to build MetaPost as a thread-safe dynamic library as well as a static executable. This will allow easy embedding inside other programs, as well as facilitating the creation of a system-level rendering service.

The two main pieces needed for this are the creation of a MetaPost instance structure to replace the current set of global variables, and the addition of an extra layer of I/O indirection.

Better numerical precision

MetaPost currently uses fixed-point arithmetic based on 32-bit integers, with the decimal dot varying between the 16th bit (for numeric values), the 20th bit (for angles), and the 28th bit (for Bézier fractions). The precision as well as the range of these calculations leaves something to be desired.

There is a web change file by Giuseppe Bilotta that uses 64-bit internal calculations instead, with the decimal dot at the 32nd, 52nd, or 60th bit. While this change file does not deal with the problems MetaPost has with ranged input data, it does solve the most obvious acute precision problems.

Storable objects

We want to add the possibility of storing and retrieving named drawing objects. These objects will be stored in memory only once, and written to the output file only once.

This will greatly reduce the memory requirements and output size for certain types of figures.

Multiple linear equation systems

All linear equations in MetaPost are part of the same equation system. With every new equation this entire system has to be updated, at a significant cost in terms of running time.

Certain macro programming styles would benefit enormously from the possibility to use disjoint sets of equations simultaneously.

Some 3D support

The ability to create and use 12-part transformations and perhaps some other operations on triplets should make it easier for macro packages to implement three-dimensional drawings.

It seems unlikely at this point that there will be true three-dimensional paths. We certainly do not object to such an extension, but both the expertise and available time to do this are sorely lacking within the current MetaPost development team.

Where to find MetaPost

- Web home page and portal:
`http://www.tug.org/metapost`
- User mailing list:
`http://www.tug.org/mailman/listinfo/metapost`
- Development & sources:
`https://foundry.supelec.fr/projects/metapost`

Managing a network TEX installation under Windows

SIEP KROONENBERG
Rijksuniversiteit Groningen
Faculteit der Economische Wetenschappen
Postbus 800, 9700 AV Groningen, The Netherlands
`siepo (at) cybercomm dot nl`

Keywords
MiKTEX, TeXnicCenter, filename database, registry, graphic file formats

This paper is about the MiKTEX installation I maintain for the Economics Department of the Rijksuniversiteit Groningen. We have long been the home of 4TEX. But when development of that project stopped, the time came that this TEX installation had to be replaced by something else. This something else was going to be MiKTEX with TeXnicCenter as editor and front end (see fig. 1).

There are various Windows editors which support MiKTEX, *i.e.* editors which have menu items and buttons for compiling and viewing your TEX documents with MiKTEX. Configuring *e.g.* TeXnicCenter or WinEdt for MiKTEX is almost automatic. TeXnicCenter is free, both as in beer and as in speech. The MiKTEX site lists a few more free editors. LaTeX-Editor[1] and Texmaker[2] seem to have a focus similar to TeXnicCenter.

MiKTEX itself comes with a configuration program, MiKTEX Options or mo.exe,[3] which has to be started from outside the editor. Over time, the MiKTEX installation has been accumulating some add-ons, especially for handling graphics; see further on.

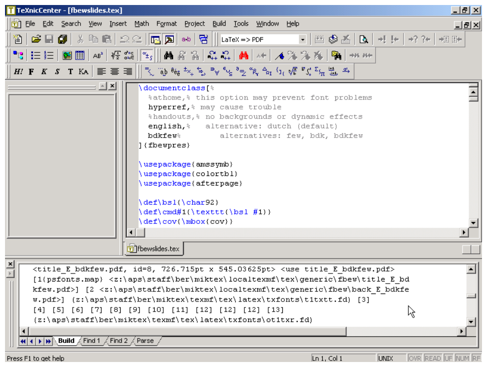
Moving from 4TEX to MiKTEX

I didn't try to create a unified 4TEX-style interface for everything, and also didn't try to replicate the functionality of 4TEX, but I did collect the local macros, fonts and graphics from 4TEX which were still in use and put them into the MiKTEX installation, sometimes with some minor tweaks.

[1]`http://www.ntu.edu.sg/home5/pg03053527/latexeditor/`

[2]`http://www.xm1math.net/texmaker/`

[3]MiKTEX also has a package manager. But of course that is not useful to users who don't have write access to the installation.


TeXnicCenter - [fbewslides.tex]
File Edit Search View Insert Math Format Project Build Tools Window Help
LaTeX => PDF
fbewslides.tex

```
\documentclass[%
  %athome,% this option may prevent font problems
  hyperref,% may cause trouble
  %handouts,% no backgrounds or dynamic effects
  english,%   alternative: dutch (default)
  bdkfew%        alternatives: few, bdk, bdkfew
]{fbewpres}

\usepackage{amssymb}
\usepackage{colortbl}
\usepackage{afterpage}

\def\bsl{\char92}
\def\cmd#1{\texttt{\bsl #1}}
\def\cov{\mbox{cov}}
```

```
<title_E_bdkfew.pdf, id=8, 726.715pt x 545.03625pt> <use title_E_bdkfew.pdf>
[1{psfonts.map} <z:\aps\staff\ber\miktex\localtexmf\tex\generic\fbew\title_E_bd
kfew.pdf>] [2 <z:\aps\staff\ber\miktex\localtexmf\tex\generic\fbew\back_E_bdkfe
w.pdf>] (z:\aps\staff\ber\miktex\texmf\tex\latex\txfonts\t1txtt.fd) [3]
[4] [5] [6] [7] [8] [9] [10] [11] [12] [12] [12] [13]
(z:\aps\staff\ber\miktex\texmf\tex\latex\txfonts\ot1txr.fd)
```

Figure 1: TeXnicCenter, a MiKTEX front end.

I dropped support for the old LATEX 2.09 since it would have meant real work for something that might not even get used.

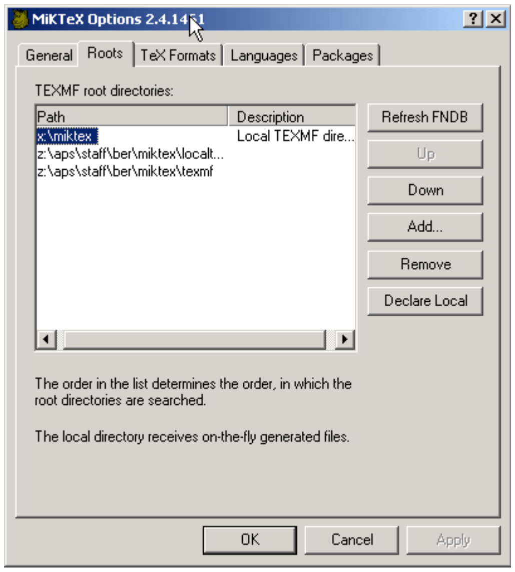
The MiKTEX installation was put online early in 2003. For a year and a half, MiKTEX and 4TEX were available side by side, but in the end, after six months notice, I removed 4TEX from the network.

Layout and contents of the installation

Texmf trees

As to the organization of macros, fonts and other support files, MiKTEX is rather similar to other modern free TEX implementations: it organizes its files into several *texmf trees*, which have a standardized structure: *e.g.* font-related files are in subdirectories `fonts\tfm`, `fonts\type1`, etc., and LATEX macros are in `tex\latex`. Each tree follows such a structure and has its own filename database. Users can configure in which order the trees are going to be searched.

I configured the following three trees (fig. 2): a main tree for files coming from the MiKTEX distribution; a department tree for local additions, includ-



TEXMF root directories:

Path	Description
x:\miktex	Local TEXMF dire...
z:\aps\staff\ber\miktex\localt...	
z:\aps\staff\ber\miktex\texmf	

The order in the list determines the order, in which the root directories are searched.

The local directory receives on-the-fly generated files.

Figure 2: Defining texmf trees and their priorities with MiKTEX options.

ing the files inherited from the 4TEX installation; and a user tree for people's own macros and other files. Users have write access only to their own user tree. With this setup, a MiKTEX upgrade won't interfere with the department tree, and anything done to the network installation leaves user files alone.

Package selection

MiKTEX is distributed as a setup program and a set of packages. The MiKTEX Setup Wizard lets you choose between three initial package sets: small, large or everything, which you can modify later on. I picked the 'large' package set, and added and removed some packages afterwards.

Add-ons

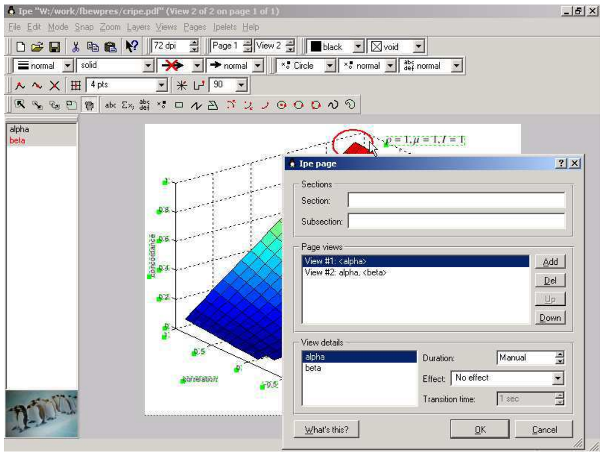
Add-ons to MiKTEX originally included GSview and Ghostscript, but for the current edition, this was no longer necessary, since these programs were already installed separately.

For better scripting, I included the Perl `.exe` and `.dll` files, but placed them outside the search path. If people have a need for Perl then they can install their own copy, without these two files getting in the way.

Graphics support

Drawing programs

Our MiKTEX installation includes a couple of drawing programs. One of these is Ipe (`http://ipe.compgeom.org`), which has a few interesting features (fig. 3):



Sections

Section:

Subsection:

Page views

View #1: <alpha>

View #2: alpha, <beta>

View details

alpha

beta

Duration: Manual

Effect: No effect

Transition time: 1 sec

Figure 3: The Ipe drawing program has views or pages which are really assemblies of 'layers'.

- It uses pdflatex in the background for typesetting text elements. You can tune typographic details with LATEX preamble commands.
- It can import arbitrary pdf via a separate conversion utility.
- A drawing can be layered in the sense that it can be displayed incrementally in a pdf presentation. In fact, Ipe also advertises itself as a tool for making presentations.

A second drawing program is LATEXCAD, which generates LATEX picture environments. It is very old and basic and is only included for people who still have old LATEXCAD drawings in use.

Converters

I also added some PostScript, EPS and PDF conversion scripts, with desktop shortcuts which can be used as drag-and-drop targets. For conversion from pdf to PostScript I added the xpdf utilities.

I plan to write a basic GUI tool, custom tailored for our installation, which offers all available conversions from a single interface. Of course, the real work will be done by Ghostscript and other trusty command-line standbys.

There is also an installer for wmf2eps. This program offers a fairly practical way to make graphics from MS Office and other Windows programs available to LATEX. It seems that not much has happened with it lately, but it still works well enough. It relies on a virtual printer driver, and therefore isn't a good candidate for a network install. Its main advantage over simply printing to an EPS file is that it calculates a tight bounding box, rather than just turning an entire page into a graphic.

Recent versions of Ghostscript have a 'bounding-box' output device. There are now various scripts which use Ghostscript to fix bounding boxes. The as yet nonexistent GUI tool mentioned above should also offer such an option, as an alternative to using wmf2eps.[4]

An installer

Installation of the network version involves:

- storing configuration information in the registry
- creating the user tree if it doesn't exist
- creating desktop and start menu shortcuts
- generating the filename database

Because our desktop systems are very standardized, none of this requires user input.

Filename database

MiKTEX's texmf trees have an important and annoying difference with the more unixy varieties: the filename database of a tree is not stored with that tree, but in a designated 'local' tree which receives generated files. This local tree can only be the user tree. That means that it is up to the user to update his filename database if items are added to or moved around in the global installation.

The filename database can be generated from the MiKTEX Options program but, fortunately for writers of installation scripts, it can also be done from the command line:

```
initexmf --update-fndb
```

It happens often enough that the installation fails because the user becomes impatient with the generation of the filename database. Without the help of this database, MiKTEX becomes very very slow. To minimize the problem, I saved filename database generation till last in the installation process and preceded it with dire warnings about not interrupting the installer. If these warnings are ignored, then the filename database can still be generated after the fact from the MiKTEX Options program. An alternative would be to copy a pregenerated filename database to the right place during installation.

Dealing with the registry

Under Windows, almost all configuration information is stored in the registry.

The registry contains a set of hierarchically organized keys. There are several root keys. The most important ones are `HKEY_CURRENT_USER` and `HKEY_LOCAL_MACHINE`, `HKCU` and `HKLM` in short. The `HKCU` part of the registry may be on a network drive. `HKLM` is normally in a subdirectory of the Windows directory.

The actual information is contained in *values* under those keys. Values are name-data pairs.

For users, there are very few reasons to edit the registry directly. There are almost always specialized menu entries and dialogs available, such as Tools menus and Control Panel entries.

Registry tools

If you do need to view or manipulate the registry directly, Windows has a number of tools: you can browse the registry with regedit;[5] and you can export and import registry keys to and from `.reg` files with either regedit or the command-line tool reg.exe. These `.reg` files are editable text files. reg.exe may not be installed by default, though. Type '`reg /?`' for help.

Various programming languages, including Perl, VBScript and installer programs, also have functions for accessing the registry.

Finding out what registry values are needed

There are a number of techniques for identifying the registry settings that you need: one way would be to inspect the source code of the original installer.

A second method is browsing and searching the registry for likely strings, and then testing whether you have captured enough to make the program work as intended. However, such testing can be time-consuming since you sometimes have to re-login or reboot before the changes take effect.

A third method is to export the registry to a text file before installation and after installation or first use, and compare the differences. There exist auto-

[4]There is also a Linux program called wmf2eps, but I have had mixed results with it. It seems better to convert wmf and emf graphics to EPS or PDF on the original system *before* trying to use them elsewhere.

[5]Under earlier versions of Windows, regedit and regedt32 each could do things that the other couldn't. Under Windows XP, regedit combines the functionality of the earlier regedit and regedt32. Its version of regedt32 simply starts up regedit.

mated installers which do just this, but the GNU diff program works just fine.

You still have to decide which differences matter. There will probably a lot of spurious differences. For example, most programs record window positions and most recently used files in the registry.

Also, some information which occurs only once can appear to occur multiple times. In particular, under Windows 2000 and later, the keys under `HKEY_CLASSES_ROOT` (`HKCR`) are copies of keys from under `HKLM\software\classes` and `HKCU\software\classes`.

All users or not: `HKLM` *vs.* `HKCU`

Often, when you install software, there is a choice whether or not to install for all users. If you do, keys are added under `HKLM\software`; otherwise, under `HKCU\software`.

Where to look

The most important settings are put in `software\`⟨*program*⟩ and `software\classes` (either from `HKLM` or from `HKCU`). The keys under `classes` define file types and define what happens when you double-click a file in Windows Explorer. Command-line programs may not need any entries here.

Uninstall information can be found under `HKLM\software\Microsoft\Windows\CurrentVersion\Uninstall`. MiKTEX doesn't have an uninstaller yet.

Registry entries for MiKTEX itself

MiKTEX makes very modest use of the registry. It just records the locations of the texmf trees, stores uninstall information, and defines the `.dvi` file type, associating it with the yap previewer.

I also added the MiKTEX binaries directory to the search path, for those people who prefer to run MiKTEX from the command line. On Windows 2000 and Windows XP the search path and other such environment variables are stored in the registry; under `HKCU\software\environment` for the current user, and `HKLM\system\currentcontrolset\control\session manager\environment` for the local system.

Registry entries for Ghostscript and GSview

Ghostscript needs to record the location of the main `.dll` and of its own fonts and support files. GSview defines the `.eps` and `.ps` file types and associates them with itself.

Registry entries for TeXnicCenter

TeXnicCenter stores a lot of information in the registry, but it can configure itself when it is started for the first time if it can find the MiKTEX, Acrobat, Ghostscript, and GSview registry settings. All the user has to do is to answer 'yes' a few times. I decided to leave configuration of TeXnicCenter to itself.

It is possible to rerun the TeXnicCenter configuration wizard at a later date. This may come in handy whenever MiKTEX or Ghostscript or GSview has moved, or Adobe Reader has been upgraded.

It would be nice for TeXnicCenter to check at startup whether these programs are still at their previous location.

More installers

I started out with one installer, but now there are several.

A network installation for a LATEX course

A second network installation was needed for a computer course for econometrics students. This installation is a slightly stripped-down version of the first one: no department tree, and without some of the add-ons.

A cd installation

Earlier, I had already made a cd with the standard installers for MiKTEX, Ghostscript, GSview and TeXnicCenter, and a copy of the department tree, plus a file with instructions how to put everything together.

There were two problems with this: it was complicated enough that some people preferred to let me install MiKTEX for them, and other people figured that they might as well download and install MiKTEX directly from the Internet. Which was not exactly wrong, but differences in their setup sometimes made it difficult to debug their problems.

So I hope that the new installer cd has persuaded some people to avoid the do-it-yourself install.

The differences

I already listed some of the differences between staff and student installations. Some differences between the network and cd installers are:

- With the cd installer, users can choose locations for the main installation and for their own data. These locations are fixed in the network version.

- The cd installer has to copy everything to the hard disk, whereas in the network version everything is already in place. In fact, re-running the network installer is no big deal. Which is just as well, since time and again configurations get messed up by a malfunctioning network or other mishaps.

- The cd installer tests for Ghostscript and GSview. If they are missing, the user first has to install these programs, *e.g.* with the installers provided on the cd. The network version simply knows that Ghostscript and GSview are present and where they are.

- The cd does an 'All users' install; the network version doesn't. Since on our network the `HKCU` part of the registry and the start menu are on the user's network drive, you can run MiKTEX from any workstation on the network.

- The cd only contains MiKTEX fonts, not the additional department fonts. Adding fonts in MiKTEX 2.4 is tricky at best. Adding them to systems that I didn't control caused too much grief.

It was not difficult to create the installer script as one main script with four different wrapper scripts.

I kept the installation and the installer on a Linux Samba server. I managed to put all 'real' files in a single directory tree, and to access these files through four different sets of symlinks. This prevented worries about keeping the versions in sync.

Installer programs

The standard way to distribute applications at our university is to create entries in NAL, or Novell Application Launcher, using Novell ZENworks. As I understand it, ZEN identifies file system and registry differences before and after installation. With ZEN, an installer can make system changes for which the user wouldn't have permissions without ZEN. However, a first attempt to create such a NAL entry for MiKTEX, done together with our NAL specialist, was not exactly smooth sailing.

I needed something that I could develop and test on my own system. This was even more important for the student install for the LATEX course, where I had to do everything through intermediaries who weren't even in the same building.

In the first edition, which didn't include a cd counterpart, I could make do with a batch file with some embedded Perl code[6] for manipulating the search path.

The cd version of the second edition required user interaction, first for telling users to install Ghostscript if it wasn't found, second for asking users where MiKTEX should be installed. So it was really time to switch to a GUI installer.

I picked NSIS.[7] It is completely scriptable and can be used from the command line.[8] It has functions for reading and writing the registry and for creating shortcuts. It offers string handling and conditionals. You can choose to what extent you want to package files into the installer itself, *i.e.* you can also tell the installer to copy files straight from the installation media to the target system.

The principal drawback of NSIS is very low-level string handling, which is quite painful if you are used to Perl string handling and regular expressions.

I have also heard good things about InnoSetup (`http://www.jrsoftware.org/isinfo.php`), but by then I was almost finished with my NSIS installer.

Development and testing

Virtual machines

Nowadays, you don't need physical test machines any more; with software such as VMware you can create virtual guest machines for testing inside a host, *e.g.* inside your everyday PC. The hard disk of this guest computer is a very large file on the host's disk. Its screen can take up the entire physical screen, but it can also run inside a window, whatever is convenient. If host and guest have similar processors, performance can be quite decent. VMware supports Windows and Linux hosts.

There are other options for virtual machines, both commercial and free: the Xen project (`xensource.com`) is getting a lot of attention, and may in time become a very interesting alternative. See also QEMU (`qemu.org`), Win4Lin (`win4lin.com`) and Virtual PC

[6] That is, the batchfile calls Perl with the `-x` switch and itself as parameter; see the perlrun man page.

[7] `http://nsis.sourceforge.net/`

[8] For GUI addicts, there is an interface with some buttons to push. There are also third-party editors with a GUI for building dialog boxes.

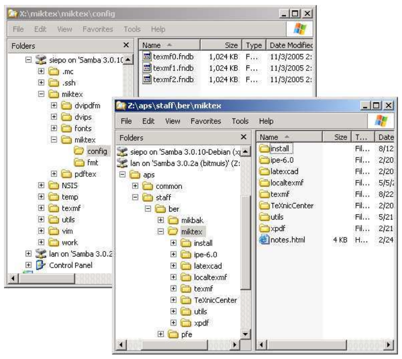


Figure 4: The Samba shares seen from the Windows client machine. It doesn't make a difference whether the Windows machine is physical or virtual.

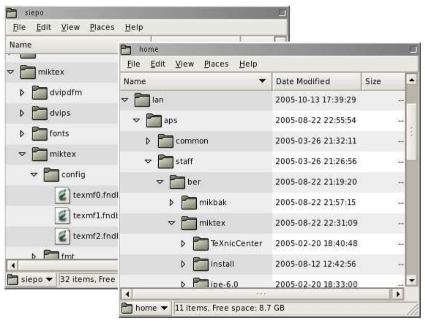

siepo
home
File Edit View Places Help
Name
Date Modified
Size
miktex
dvipdfm
dvips
fonts
config
texmf0.fndb
texmf1.fndb
texmf2.fndb
fmt
lan 2005-10-13 17:39:29
aps 2005-08-22 22:55:54
common 2005-03-26 21:32:11
staff 2005-03-26 21:26:56
ber 2005-08-22 21:19:20
mikbak 2005-08-22 21:57:15
miktex 2005-08-22 22:31:09
TeXnicCenter 2005-02-20 18:40:48
install 2005-08-12 12:42:56
ipe-6.0 2005-02-20 18:33:00
siepo 32 items, Free
home 11 items, Free space: 8.7 GB

Figure 5: The Samba shares seen from the Linux server. It makes no difference whether the server is a separate machine, a VMware host or a second VMware guest machine.

(`microsoft.com`).[9] Some of these emulators are focused more on running Windows applications than on creating a realistic test environment.

For testing installers, you can create a 'clean' virtual machine with just Windows and some indispensable programs installed. Then you can run simulations on copies of this virtual machine. Getting another clean test machine is just a matter of making a fresh copy of the original one, which takes only minutes.

Virtual networking

For networking, I let VMware create what it calls a host-only network, with no direct access to an external network. This saves me the hassle of protecting virtual Windows machines against malware from the Internet. I configured the Linux host as a Samba server, with the MiKTEX installation in a Samba share, and the user's home directory in another share. (See figures 4 and 5.)

Roaming profiles

The university has started using *roaming profiles*. The idea is to place user configuration data as much as possible on their own network home drive. This includes *e.g.* users' start menus and the `HKCU` part of the registry.

With Samba, roaming profiles means configuring the (or a) Samba server as a PDC or Primary Domain Controller. This is no fun. It means creating things called machine accounts for the client machines, and painstakingly reading Samba documentation. A very helpful and funny guide was 'The Unoffical Samba HOWTO'. You can find this document via the Samba site. Its current location is `http://hr.uoregon.edu/davidrl/docs/samba.html`.

For testing, I now make clean MiKTEX-free profiles, with just the worst default Windows settings fixed, and work with copies of those clean profiles, just as I already did with guest machines.

Disappearing file types

In theory, with MiKTEX and TeXnicCenter, roaming profiles should work perfectly: there should be no need to install or configure anything on the machine itself. In practice, definitions of file types under `HKCU`, *i.e.* all keys and values under `HKCU\software\classes`, got lost in between logins — in real life, not in my test setup.

For staff network installations, a workaround is to duplicate the file type definitions in `HKLM\software\classes`. For student network installations, students don't have write permission on `HKLM` keys, so this would only be possible with something like ZEN. As a hacky alternative, I put together a registry patch with file type definitions, *i.e.* a `.reg` file with the registry keys and values under the `HKCU` key which define these file types. By double-clicking this file, these keys and values are imported into the registry. Students have to run this registry patch before each MiKTEX session.

Next time around, if the disappearing file types problem still isn't solved, I might have to do something with ZEN after all.

[9]Virtual PC was bought from Connectix by Microsoft in the second half of 2003. The Macintosh version of Virtual PC was at the time the only real option for running Windows on the Mac. I, along with many other Mac owners, was duly shocked by this sell-out. But in the meantime, other emulators appeared, and now that the Mac is moving to Intel, we can hope for VMware-quality virtual machines on Mac OS X from other companies than Microsoft.

Typography-based on-line help for TEX[*]

GYÖNGYI BUJDOSÓ
Faculty of Computer Science
University of Debrecen
H-4010 Debrecen, P.O.B. 12, Hungary
`bujdoso (at) inf dot unideb dot hu`

Abstract

People using TEX often search for on-line information about TEX. Although many on-line systems show the syntax of many commands and environments, few or none contain typographical recommendations for them. For example, we can find the command `\underline` *and its syntax but there is no hint that underlining texts is not recommended in documents even if the text is a section title.*

Our project was to begin developing a typography-based TEX help system. This presentation deals with the main features of the system, how to integrate important typographical recommendations, source code of TEX commands and environments, designed forms and layouts, and the most problematic grammatical rules.

Introduction

When people use TEX they sometimes need help with command syntaxes. There are some very good web based systems about how to use TEX (e.g., [5, 7]) which assist people in how to use a command or environment, how to set the indentation of a paragraph, how to italicize a word or how to modify the labels of an enumeration. It is quite useful, saves a lot of time, but is it enough? Is it possible to find information on the web on how big the indentation should be, which words should be in italics or bold, what types of labels should be applied?

It is safe to say that it is hard to find information on typographical recommendations on the web and most people do not acquire books on typography.

An idea follows from the foregoing: an on-line system which contains more than the technical details for typesetting should be offered to those interested.

Basic considerations

The curriculum in many courses on computer science for beginners (including primary and secondary schools) contains word processing. In general, this means teaching techniques on how to use menus, dialog boxes and icons. The methodology is similar when teaching TEX (for example, for students in mathematical specialties). This method of teaching word processing results in students being able to use some functions of a word processor or TEX, but they do not care about the layout of documents. Usually they use the default settings of one (and only one) style file, and bold letters for emphasizing in-line texts, sometimes they use headings or centering — and that is all. The layout of the documents is not harmonious and not aesthetic, sometimes quite structureless.

The first idea was to give information to students on typography when teaching techniques of word processing. This method was not successful enough, because students thought that these rules and recommendations were without importance. Shifting the focus from teaching techniques to teaching typography [2] was more successful, because information describing a form was about typography first, then followed by technical issues. The students were more motivated and attended to the layout of their documents.

Currently many students at universities do not have time to attend courses on word processing, and have no money for buying books, so they mostly use the web for getting information in this field (just as with almost everything). Nevertheless, on-line help for word processing generally contains only technical information. How to motivate students and, for that matter, anyone to give more consideration to layout?

A solution could be an on-line help system based on typography [3].

[*]Supported by DIP Kooperációs Kutató Központ, University of Debrecen, project no. GVOP-3.2.2.-2004-07-0021/3.0.

Versions, concepts and problems

The *first version* of our "on-line help" was more of a textbook on the web than on-line help. It was a static system of web pages on certain parts of the curriculum, namely a set of

- sample pages
  - containing the description of some technique,
  - showing a recommended layout of the form in question, and which
  - has to be reproduced.

The system had two main problems: The most important problem was that the sample pages did not contain the entire source code of the special forms, only the syntax and the detailed descriptions of the commands (because students had to reproduce the pages). There was a time when this type of learning was motivating for our students, but the world has changed; nowadays people prefer ready-to-use things. The second problem was that it was hard to find a command or anything else in the curriculum. It was time to change the concept.

The *second version* was a system of web pages. These pages contained the same topics of the first version, plus

- the source code of many forms.

A new structure was applied: small pages with less information on each and with many links to the related topics and forms.

Some problems arose when more topics and text had to be inserted into the system. The descriptions had to be sliced and organized in another way.

The result: hundreds of small web pages (containing much redundant information), hundreds of pictures and thousands of links. After a while, it was impossible to handle and update the system. The system collapsed before being published! Thus the concept had to be changed.

New techniques

Having worked with systems which were not efficient enough, we planned to develop a free system which could be a help to teachers (with or without programming skills) in creating and organizing thousands of files of their curricula. This system was designed to be different from the Learning Content Management Systems (lcms) which are available free of charge. (A free lcms needs a system administrator for installing, maintaining and updating it.) This project has been canceled due to various reasons.

The frame that we can use can be a free lcms. Among others, Moodle [6] seems to be the best frame for our purposes, as its language support is quite good and it has several agreeable features and other functions.

Moodle — like lcms frames in general — has many advantages such as tests, logs, chats, possible tutoring, white boards, searching, etc. Nevertheless it has some disadvantages, too. One of its best features, i.e., its uniformity (that is, each window dialog box, etc. has the same layout, as determined by the chosen skin) is considered to be quite disadvantageous from the current perspective. Another problem is that it lacks some functions that could make the system easier to deal with. It is too rigid and hard to personalize.

Formats

One of the biggest problems with using on-line systems is that people need hardware and web access to get any information. For people who work always on computers, of course this does not cause any problem because they have both. For other people, who have neither laptops in their bags nor free (nor unlimited) Internet access, getting on-line information can be problematic.

A more convenient way would be if the system can be downloaded and installed if needed (in an easy way) onto a local computer.

Also, visualization is a common learning technique. The system should support this method of study by offering a printable version of the needed part(s) of the curriculum to the users. Such pieces of information should be at anyone's disposal.

Accessibility

Accessibility is an important question in typesetting texts, too. Many disabled people use TEX, so we must help them, one major group being people with visual impairment. Initiatives can be read, e.g., at [8, 9, 11], or in Hungarian at [1]. Many useful recommendations on interactive design can also be read, e.g., at [4] or [10]. Most of them should be adapted to the new on-line help system.

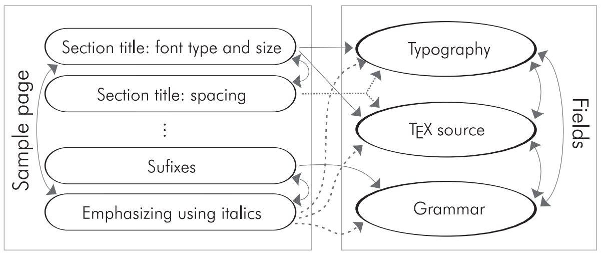


Figure 1: Fields and relations

The minimal requirements for our on-line help system are the following:

- It must contain functions for enlarging text, menu items, icons, pictures, etc. These would help people with visual as well as motor impairments.
- Alternative text should be assigned to each picture.
- It should offer possibilities to users for setting colors and contrast applied by the system to the users' desire. This feature is very important, for example, for color- and night-blind people.

Augmented content

Some new topics and many new (cross-)references (see Figure 1) have to be built in. The following new fields are planned to be embedded into the on-line help:

- typographical recommendations,
- commonly (not) used rules of grammar,
- a class/style file (or a simple macro file) maker to LaTeX and plain TeX.

Also, some new features have to be added and many relations should be highlighted:

- links from special forms of a displayed page to typographical descriptions and samples that are concerned to the forms (see Figure 2),
- links from special forms to their source code,
- links to grammatical rules of problematic words, suffixes, etc.,
- supporting different languages,
- demonstration of designed layouts,
- representative samples on good and bad forms and usage of grammatical rules.

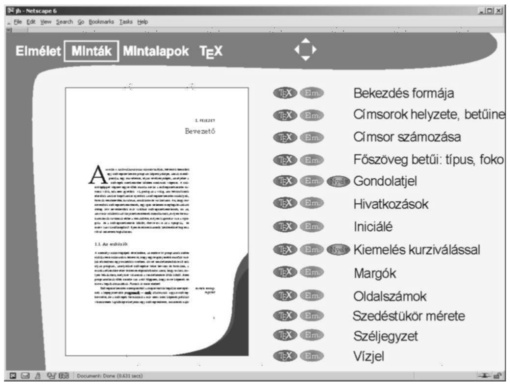


Figure 2: Sample page and relations

About the project

The project consists of two main parts: developing the content, building up the frame and aligning it to the needs of the content and users.

The content on TeX is under development; many of the necessary topics and parts are being gathered. The contained descriptions have to be organized into self-contained pieces by carefully abolishing the redundancies, meaning that a system of learning objects has to be developed.

There are many texts on typography, but we need to enlist a typographer who would write more, or at least referee them.

The most problematic part is building up and aligning the frame. It needs programmers who have to be supported. Trying to find financial support is in progress.

Conclusions

Our main goal is to develop a system that can motivate people to create good layouts in their work. Developing an easy-to-use on-line help system on word processing combined with typographical issues is a difficult task. It requires a lot of work from several people on various fields: instruction, typography, human–computer interaction, system programming.

In case we succeed in developing this on-line system, it will be a good frame to adapt the content to other languages, and offer a place on the web where people can get information not only about TeX, but typesetting texts and designing aesthetic layouts.

References

[1] *Accessibility initiatives and principles,* (in Hungarian), `http://vmek.oszk.hu/vmek2/ajanlas.phtml` `http://www.paramedia.hu/` `http://weblabor.hu/cikkek/iranyelvek` `http://www.w3c.hu/forditasok/wai_quick_tips.html`.

[2] Gyöngyi Bujdosó, *Teaching word-processing at our university*, Proceedings of Informatika a felsőoktatásban '96 (August 27–30, 1996, Debrecen, Hungary), 1996, pp. 101–109, (in Hungarian), `http://www.iif.hu/rendezvenyek/networkshop/96/vegl.html`.

[3] Gyöngyi Bujdosó, *Online learning course on word processing based on typography*, Proc. of EMES 2005 (May 26–18, 2005, Oradea, Romania), in: Analele Universităţii din Oradea, 2005, pp. 33–36.

[4] *What accessibility means*, The gnome Project, 2005, `http://developer.gnome.org/projects/gap/access-def.html`.

[5] W. Macewicz and S. Wawrykiewicz, *Wirtualna TEX akademia*, 2004, (in Polish), `http://www.ia.pw.edu.pl/~wujek/tex/`.

[6] *Moodle: A modular object-oriented dynamic learning environment*, `http://moodle.org`.

[7] *(LA)TEX Navigator, A (LA)TEX encyclopaedia*, `http://tex.loria.fr/`.

[8] *Section 508*, `http://www.section508.gov/`.

[9] *SENDA: Special Educational Needs and Disability Act*, `http://www.ukcle.ac.uk/directions/issue4/senda.html`.

[10] Bruce Tognazzini, *First principles of interaction design, in: Interaction design solutions for the real world*, AskTog, 2003, `http://asktog.org`.

[11] *WAI: Web Accessibility Initiative*, `http://www.w3.org/WAI/`.

Automatic non-standard hyphenation in OpenOffice.org

LÁSZLÓ NÉMETH
nemeth (at) openoffice dot org

Abstract

The hyphenation algorithm of OpenOffice.org 2.0.2 is a generalization of TEX's hyphenation algorithm that allows automatic non-standard hyphenation by competing standard and non-standard hyphenation patterns. With the suggested integration of linguistic tools for compound decomposition and word sense disambiguation, this algorithm would be able to do also more precise non-standard and standard hyphenation for several languages.

Introduction

Standard hyphenation consists of splitting a word and including a hyphen at the end of the first part of the split word (unless the word already contained a hyphen or n-dash at the break). While standard hyphenation is widely applicable, several languages also use non-standard hyphenation.

Table 1 shows examples for non-standard hyphenation: character deletions and other changes at hyphenation break points in European writing systems. Some non-standard hyphenation can be handled easily by computer, like the mandatory middle dot deletion from Catalan digraph *l·l*. But complex analysis is necessary for languages, like German,[1] Hungarian, Swedish and Norwegian to recognize hyphenation points. For instance, the Swedish word form *glassko* has three different meanings, and can be hyphenated as *glas- sko* (glass shoe), *glass- ko* (ice cream cow) and in the non-standard way, *glass- sko* (ice cream shoe).

Such non-standard hyphenation plays an important role in good typesetting. Commercial DTP programs, even word processors, support automatic non-standard hyphenation, often by licensing third party libraries. The most important free alternatives, such as Apache FOP, GNU Troff, KDE KOffice, OpenOffice.org, Scribus, and TEX and its variants, do not support automatic non-standard hyphenation. TEX has a hyphenation primitive, the `\discretionary` command. There are TEX macros in the Babel package for non-standard hyphenation, for instance, `\lgem` for Catalan *l·l*, `\ck` or `"ck` for German, `~ssz` for Hungarian, `\=` for Polish, but there is no real automatic non-standard hyphenation in TEX. Omega 2 has promising developments towards implementing sophisticated automatic non-standard hyphenation for German and other languages [4, 5].

The aim of the present project was to implement language-independent automatic non-standard hyphenation in OpenOffice.org. In this article we present our results, introduce old and new hyphenation algorithms, extension of the Hungarian hyphenation patterns and finally show the possibility of integrating compound word decomposition and word sense disambiguation to our algorithm.

Results

TEX's hyphenation is the de facto standard in the free software world, because the hyphenators of the free programs mentioned in the previous section are all based on Liang's hyphenation algorithm from TEX82 [9], and use the TEX hyphenation patterns. Thus, we have developed an extension for OpenOffice.org's ALT Linux LibHnj hyphenator to do automatic non-standard hyphenation. The result is based on a generalization of Liang's original algorithm which also allows easy integration of special linguistic tools to handle compound word decomposition and word sense disambiguation in automatic hyphenation. The Hungarian hyphenation patterns were extended with non-standard hyphenation patterns.

The improved hyphenation library (without integrated linguistic tools) is part of the OpenOffice.org 2.0.2 with the extended Hungarian hyphenation patterns. Developers can download a standalone version

[1]German orthography before the spelling reform of 1996.

Language	Example	Hyphenation	Description
Catalan	*paral·lel*	*paral- lel*	digraph *l·l* represents long (geminated) *l*
Dutch	*reëel*	*re- eel*	diaeresis and hyphenation sign syllable breaks
	omaatje	*oma- tje*	vowel lengthening with diminutive *-tje*
English	*eighteen*	*eight- teen*	suggested pretty hyphenation by D. E. Knuth [6]
German	*Zucker*	*Zuk- ker*	digraphs *ck* and *kk* represent long *k*
	Schiffahrt	*Schiff- fahrt*	triple consonants at compound word boundary
Greek	Μαΐου	Μα- ΐου	diaeresis and hyphenation sign syllable breaks
Hungarian	*asszonnyal*	*asz- szony- nyal*	simplified double digraphs (long *sz* and *ny* phonemes)
Norwegian	*bussjåfør*	*buss- sjåfør*	triple consonants at compound word boundary
Polish	*kong-fu*	*kong- -fu*	repeated hyphen at line begin
Swedish	*tillåta*	*till- låta*	triple consonants at compound word boundary

Table 1: Non-standard hyphenation in European languages

```
. a l g o r i t h m .
   4l1g4
     l g o3
       1g o
             2i t h
                 4h1m
  -----------------
   4 1 4 3 2 0 4 1
  a l-g o-r i t h-m
```

Figure 1: TeX hyphenation of 'algorithm'

of this library with an example executable from the Lingucomponent project home [14].

Liang's hyphenation algorithm

Franklin M. Liang's hyphenation algorithm is based on competing hyphenation patterns. The patterns can give excellent compression for a hyphenation dictionary, and using these patterns the fast hyphenator algorithm can also correctly hyphenate unknown (non-dictionary) words most of the time. Liang's work covers also the machine learning of the hyphenation patterns and exceptions by PatGen pattern generator.

The hyphenation patterns can allow and prohibit hyphenation breaks on multiple levels. Figure 1 shows the pattern matching on the word 'algorithm'. The TeX English hyphenation patterns `4l1g4`, `lgo3`, `1go`, `2ith` and `4h1m` match this word and determine its hyphenation. Only odd numbers mean hyphenation breaks. If two (or more) patterns have numbers in the same place, the highest number wins. The *al- go- rith- m* hyphenation is bad, but the last one-letter hyphenation is suppressed by TeX, so we end up with the correct *al- go- rithm*.

One of the most notable features of this pattern-based hyphenation is the human-readable format of the knowledge database, in contrast to an equivalent finite state machine or a similarly good artificial neural network. This format is good for manual checking and corrections.

Missing features

In TeX's automatic hyphenation the most wanted features are non-standard hyphenation, compound word analysis, word sense disambiguation and taboo word filtering [12, 13].[2]

Sojka's non-standard hyphenation extension

In [12] Petr Sojka suggests a non-standard hyphenation extension for Liang's algorithm. His algorithm first searches all hyphenation points of a word using Liang's algorithm, and then matches patterns from a non-standard hyphenation table at valid hyphenation points, replaces the matching pattern with a special character, and rechecks the hyphenation of the new word at this special character with Liang's algorithm. The non-standard hyphenation point will be chosen if the second hyphenation is successful. Using a *ck→% (k- k)* pattern data from a non-standard hyphenation table, the German word *Zucker* will be *Zu%er* after

[2]Liang's hyphenation algorithm and its compact implementation using packed trie data structure was perfect twenty-five years ago for English and for computers with less than a few MB RAM. Nowadays internationalization (handling multiple languages) is a standard in software industry and free software development. Modern personal computers have much more memory and speed to enable using additional special linguistic tools in hyphenation.

Pattern	Example	Hyphenation
`l·1l/l=l`	*paral·lel*	*paral- lel*
`e1ë/e=e`	*reëel*	*re- eel*
`a1atje./a=t,1,3`	*omaatje*	*oma- tje*
`eigh1tee/t=t,5,1`	*eighteen*	*eight- teen*
`c1k/k=k`	*Zucker*	*Zuk- ker*
`schif1f/ff=f,5,2`	*Schiffahrt*	*Schiff- fahrt*
1ΐ/=ί	Μαΐου	Μα- ίου
`s1sz/sz=sz`	*asszonnyal*	*asz- szony-*
`n1ny/ny=ny`		*nyal*
`bus1s/ss=s,3,2`	*bussjåfør*	*buss- sjåfør*
`7-/=-`	*kong-fu*	*kong- -fu*
`.til1lå/ll=l,3,2`	*tillåta*	*till- låta*

Table 2: Extended hyphenation patterns for Table 1

the replacement, and the pattern `zu%1er` permits non-standard hyphenation with *k- k* (*Zuk- ker*).

Problems

It's possible to use the pattern generator on a prepared input dictionary for Sojka's algorithm, but then we lose the human-readable format of hyphenation patterns. The biggest problem is to use competing patterns on multiple levels. That is why instead of using difficult redundant patterns with special hyphenation characters, Sojka suggests global parameters (left and right non-standard hyphenation penalties) to forbid standard hyphenations near the non-standard hyphenation points. But German, Hungarian, Norwegian and Swedish non-standard hyphenation need true competing patterns.

OpenOffice.org's extension

To keep the flexibility of Liang's algorithm, OpenOffice.org augments the original hyphenation patterns with extended patterns defining non-standard hyphenation points as subregions and replacements of the subregions. To keep the clear syntax, a non-standard hyphenation pattern is denoted as a plain hyphenation pattern and a record separated by a slash.

For example, the pattern `zuc1ker/k=k,3,2` represents the hyphenation of *Zucker*. This means the non-standard hyphenation subregion will be replaced by `k=k`, where the `=` indicates the break point with a hyphen. The subregion begins at the 3rd character, and contains 2 characters (*ck*).

```
. a s s z o n n y a l .
    s1s z/sz=sz,1,3
            n1n y/ny=ny,1,3
  ------------------
   0 1 0 0 0 1 0 0 0/sz=sz,2,3,ny=ny,6,3
  a s-s z o n-n y a l/sz=sz,2,3,ny=ny,6,3
```

Figure 2: Hyphenation of *asszonnyal*

Table 2 shows possible hyphenation patterns for Table 1. The dots in the patterns match the word boundaries. The first dot doesn't affect the character positions in the non-standard hyphenation subranges: `.zuc1ker/k=k,3,2`. Figure 2 shows the result of applying multiple non-standard pattern matching.

Rules

A single subregion must contain exactly one hyphenation point (indicated by an odd number in Liang's syntax). There may also be explicit non-breakable points (indicated by even numbers) in the subregion, and any breakable or non-breakable points out of the subregion.

A standard and a non-standard hyphenation pattern matching the same hyphenation point must not be on the same hyphenation level. For instance, `c1` and `zuc1ker/k=k,3,2` are invalid, while `c1` and `zuc3ker/k=k,3,2` are valid extended hyphenation patterns.

Unicode character encoding

Unicode is the basis for internationalization.[3] Thanks to the unambiguous start positions of the multibyte-characters, Liang's algorithm works perfectly with the UTF-8 Unicode encoding. Subregion parameters of non-standard hyphenation patterns use Unicode character (not byte) positions and lengths.

Changing hyphen

Missing or alternative hyphenation marks are handled by using underline characters instead of equal signs in our non-standard hyphenation patterns, where underline character indicates only the break point, without an implied hyphen. For example, using the underline with an explicit hyphen, `k-_k` and `k=k` are equivalent

[3]Not only for exotic writing systems. Affix-rich languages can combine different 8-bit character codes in one word. For example, *Nexøről* (*about Nexø* in Hungarian) contains special characters from Latin-1 and Latin-2 character tables.

Example	Hyphenation	Description
meggy	*meggy*	noun with long phoneme *gy* (*sour cherry*)
meggyez	*megy- gyez*	derived verb (*to do something with sour cherry*)
meggyíz	*meggy- íz*	compound (*sour cherry jam*)
meggyőz	*meg- győz*	verb prefix *meg-* + verb *gyõz* (*persuade*)
esszé	*esz- szé*	long phoneme *sz* (*essay*)
Jamesszé	*James- szé*	noun *James* + suffix *-szé* (*[to become] James*)
samesszé	*samesz- szé*	noun *samesz* + suffix *-szé* (*[to become] verger*)
esszék	*esz- szék*	noun *esszé* + plural *k* (*essays*)
vizesszék	*vizes- szék*	compound (special *chair* (*szék*) in Hungarian folklore)
rekesszék	*rekesz- szék*	verb *rekeszt* + suffix *-jék* (third-person plural *obstruct!*)
berekesszék	*berekesz- szék*	prefix *be* + verb *rekeszt* + suffix *-jék* (third-person plural *adjourn!*)
kirekesszék	*kirekesz- szék*	prefix *ki* + verb *rekeszt* + suffix *-jék* (third-person plural *exclude!*)
kerekesszék-	*kerekes- szék-*	compound (*wheel chair*)

Table 3: Hungarian hyphenation examples with ambiguous *ggy* and *ssz* patterns

patterns.[4] This notation is functionally equivalent to TEX's `\discretionary` command.

Extending Hungarian hyphenation patterns

The Hungarian language uses simplified forms to represent its double digraph and trigraph consonants (*sz+sz→ssz*, *dzs+dzs→ddzs*, etc.), but hyphenation undoes the simplification (*sz- sz*, *dzs- dzs*). Some ambiguity results from this non-standard hyphenation in Hungarian, caused by rich compounding and affixation, see Table 3.

Manual extension of the Huhyphn Hungarian hyphenation patterns based on Hungarian vocabularies and morphology has been accomplished, and the result contains over two thousand non-standard hyphenation patterns. For example, Figure 3 shows the competing patterns matching the word *esszé* (*essay*).

The Huhyphn distribution consists over 63 thousand hyphenation patterns generated from a 2.5 million word hyphenation dictionary by PatGen [10]. Our experience shows that with the manual extension of this database, the results are as good as the Hungarian commercial hyphenator MorphoLogic Helyesel[5]. What's more, extended Huhyphn works well on unknown words, resulting in significantly better automatic typesetting.[6]

```
. e s s z é .
     1s z é
       1z é
. e2
    s2s z
      s2z
         2é .
    s3s z é .
. e s5s z é/sz=sz,2,3
  ---------
   2 5 2 2/sz=sz,2,3
  e s-s z é/sz=sz,2,3
```

Figure 3: Non-standard hyphenation of *esszé*

Linguistic tools for better hyphenation

Pattern-based hyphenation doesn't work well on languages with an unlimited number of compound words [7]. Compound word decomposition by patterns results in an enormous number of hyphenation patterns in the Huhyphn distribution. However, within a few minutes, an expert could be able to find a dozen badly hyphenated compound words in Magyar webkorpusz, a Hungarian gigaword corpus with 21 million word forms. We need more sophisticated compound word decomposition methods, like SiSiSi [1, 7, 8]. OpenOffice.org's Hunspell spell checker also has morphological analyzer capability to decompose compound words. We suggest a simple method and formalism to integrate these tools with the pattern-based hyphen-

[4]It doesn't work in OpenOffice.org, yet!

[5]Hyphenator of Hungarian MS Office, Adobe InDesign, Adobe PageMaker and QuarkXPress.

[6]Helyesel hyphenates only known words, and it cannot handle many proper compounds, because its compound decomposition algorithm cannot decompose compounds from three or more dictionary words.

```
  g l a s s k o       g l a s s k o
. g l a s . s k o .  . g l a s s . k o .
          .7                     .7
  ---------------     ---------------
   0 0 0 0 7 0 0       0 0 0 0 0 7 0
   0 0 0 7 0 0         0 0 0 0 7 0
  g l a s-s k o       g l a s s-k o

  g l a s s k o
. g l a s . . s k o .
          .7
            .7
        s .8.9s/ss=s,1,4
  ----------------
   0 0 0 0 8 9 0 0/ss=s,4,4
   0 0 8 9 0 0/ss=s,4,2
  g l a s-s k o/ss=s,4,2
```

Figure 4: Hyphenation by decomposition

ation. Another advantage of the integration is that the external linguistic tools could also provide word sense disambiguation (for example, using part-of-speech taggers) to hyphenate the ambiguous words in hyphenation dictionaries.

Dots within patterns

Dots denote word boundaries in Liang's algorithm. Extending this formalism, let us also allow dots to denote the word boundaries within compounds. The compound word decomposition makes only a boundary annotation with dots, and we can hyphenate the decomposed word by dotted hyphenation patterns.

For instance, the Swedish word *glassko* would be *glas.sko* or *glass.ko* after compound word decomposition, and can be hyphenated with the pattern `.7` as in Figure 4.

Double dots

We denote non-standard compounding by double dots, as in *glas..sko*. This annotated word can then be hyphenated with a non-standard hyphenation pattern, such as `s.8.9s/ss=s,1,4` in our example.

The annotation is removed from the output of the hyphenation algorithm, as in the three possible annotated and hyphenated forms of *glassko* in Figure 4. With a suitable word sense disambiguation, the pattern based hyphenator is given exactly one of them. (Without word sense disambiguation, *glassko* is not annotated and hyphenated).

Conclusion

The new version of OpenOffice.org contains state-of-the-art Hungarian hyphenation, solving the problem of automatic non-standard hyphenation in a generalized way. The extended version of Liang's hyphenation algorithm is suitable for other languages. With the suggested formalism and minimal extension, the algorithm can also be integrated with sophisticated linguistic tools to handle compound word decomposition and word sense disambiguation in automatic hyphenation.

Acknowledgments

OpenOffice.org's improved hyphenation was demonstrated at the 3rd Conference on Hungarian Computational Linguistics in 2005, Szeged, Hungary, thanks to Media Education and Research Center (MOKK), University of Technology and Economics, Budapest. I would like to thank Stefan Baltzer and other developers at Sun Microsystems for testing and quick integration of the extended ALT Linux LibHnj library into OpenOffice.org, Péter Szabó for his helpful comments and Yannis Haralambous for his information about Dutch and Greek non-standard hyphenation. Also thanks to Marcin Miłkowski and the authors of various Internet sources [2, 3, 11] for the Polish and other examples. Special thanks for the editorial work of *TUGboat*.

References

[1] W. Barth and H. Nirschl. Sichere sinnentsprechende Silbentrennung fur die deutsche Sprache. In *Angewandte Informatik*, volume 4, pages 152–159, 1985.

[2] Linda Andersson et al. *Performance of Two Statistical Indexing Methods, with and without Compound-word Analysis*. `http://www.nada.kth.se/kurser/kth/2D1418/uppsatser03/LindaAndersson_compound.pdf`.

[3] Dave Fawthrop. *Hyphenation by algorithm of English/American and other languages*. `http://www.hyphenologist.co.uk/`, 2000.

[4] Yannis Haralambous. New hyphenation strategies in Omega v2. In this volume, pp. 98–103.

[5] Yannis Haralambous and Gábor Bella. Omega becomes a texteme processor. In *Actes d'EuroTEX*, pages 99–110, 2005.

[6] Donald E. Knuth. *The TEXbook*, page 314. Addison-Wesley, 1984.

[7] Gabriele Kodydek. A word analysis system for German hyphenation, full text search, and spell checking, with regard to the latest reform of German orthography. In *Text, Speech and Dialogue: Third International Workshop (TSD 2000)*, pages 39–44, 2000.

[8] Gabriele Kodydek and Martin Schönhacker. Si3trenn and Si3Silb: Using the SiSiSi word analysis system for pre-hyphenation and syllable counting in German documents.

[9] Franklin M. Liang. *Word Hy·phen·a·tion by Com·put·er*. Stanford University, 1983. http://www.tug.org/docs/liang.

[10] Bence Nagy. *Huhyphn— Magyar elválasztás TEX-hez, Scribushoz és OpenOffice.org-hoz*. http://www.tipogral.hu, 2003.

[11] Ole Michael Selberg. *Nohyphbx.tex introduction*. http://home.c2i.net/omselberg/pub/nohyphbx_intro.htm, 2005.

[12] Petr Sojka. Notes on compound word hyphenation in TEX. *TUGboat*, 16(3):290–296, September 1995.

[13] Petr Sojka and Pavel Ševeček. Hyphenation in TEX — Quo Vadis? *TUGboat*, 16(3):280–289, September 1995.

[14] Standalone version of ALT Linux LibHnj hyphenation library. *OpenOffice.org Lingucomponent project*. http://lingucomponent.openoffice.org/.

What tools do ConTEXt users have?

HANS HAGEN
Pragma ADE, Hasselt
`pragma (at) wxs dot nl`

A bit of history

When we started working on ConTEXt, MS Windows (and before that 4Dos) was our main platform; it still is for development (we use Unix on the web and file servers and the Mac for fun). So, when ConTEXt was integrated into the TEX distributions we faced the problem of portability. Since one needs auxiliary programs[1] for e.g. sorting an index, we had written TEXutil, and the lack of a commandline handler made us come up with TEXexec. Both were written in Modula but were rewritten in Perl in order to be usable on platforms other than MS Windows. It was easier to maintain a Perl version than to deal with low-level platform issues indefinitely.

As both our own and user demands grew, we wrote more tools and found out that they could best be written in Ruby. In the meantime TEXexec has been rewritten in Ruby, and relevant parts of TEXutil have been merged into it.

Launching scripts

Starting a script on an MS Windows box can be done using a so-called stub, a small program or command file with the same name that locates a similarly named script. On Unix some shell magic can be used to do the same or one can fall back on a magic preamble (a Bash/Perl mixture) fooling the operating system into locating and spawning the script using the right interpreter. By now, MS Windows has a convenient file association mechanism (but one has to activate it first) while Unix needs a (nowadays less path sensitive) shebang line and a suffixless copy of the script.

Nevertheless we decided to come up with a less sensitive approach which also gave us the opportunity to accomplish a few more things: TEXMFstart. This script locates and executes a script (or program) in the TEX tree and executes it.

```
texmfstart texexec somefile.tex
```

[1]We will use the terms ‘scripts’ and ‘programs’ interchangeably.

When you incorporate TEX in workflows, calling TEXexec this way is rather future safe. Actually, because of this method, we could make the transition from TEXexec being a Perl script to being a Ruby program without too much trouble. A side effect of this way of lunching scripts is that nested calls are faster because some information is passed on to child runs.

The script is also able to sort out a couple of things, for instance where files reside. Nowadays one will seldom use TEX alone and not all text processing (or related) programs have a clear concept of resource management and/or can work well with a TDS conforming tree.

```
texmfstart bin:xsltproc --output=new.xml \
  kpse:how.xsl old.xml
```

This[2] will locate the file `how.xsl` in the TEX tree and expand the filename to the full path. That way one can keep XSLT scripts organized as well. There are a few more such prefixes.

Other features are locating and showing documentation and launching editors with files located in the tree. The following call will open the `texmf.cnf` file that is currently used.

```
texmfstart --edit kpse:texmf.cnf
```

The script can initialize a tree so one can effectively run multiple trees in parallel. It does so by loading (when present) a file with variable specifications (more later about that).

```
texmfstart --tree=e:/tex-2003 \
  texexec somefile.tex
```

We often use a different tree for each project because commercial fonts may be project related and this way we can move a tree around without running into copyright problems (read: installing all fonts on each box).

```
texmfstart --tree=e:/tex-2003 \
  texexec somefile.tex
```

[2]The backslash at the end of line denotes a continued line.

Another handy feature is conditional processing. In the following case the test file will only be processed when it has changed.

```
texmfstart --ifchanged=test.r --direct R \
  "-q --save --restore < test.r"
```

In a similar fashion one can make running dependent on time stamp comparison. More details can be found in the manual.

Managing ConTEXt runs

The TEXexec script manages a user's TEX run. There are many factors that influence such a run:

- Since ConTEXt uses the same format for all backends, it depends on loading the relevant backend driver modules. Although one has complete control, life can be made easier when this is done automatically.
- A first pass may generate data needed in a successive pass. There may be references, tables of contents, indices, etc. so we need a way to manage multiple runs. We have to make sure that neither fewer nor more runs than needed take place.
- A run may demand further action between runs, like graphic manipulations or delayed MetaPost execution.
- We may want to run different TEX engines, apply different backends, use different user interfaces. Also, the name and way of calling TEX may change over time, something that we don't want users to be bothered with.
- We may want to process a TEX or XML file under different style regimes or enable style-specific modes.
- The document may need an additional page imposition pass, managed in such a way that no auxiliary data gets messed up.
- We may want to close and open the result in a viewer after the run is done.

This and a bit more is handled by TEXexec. When dealing with ConTEXt files the script will do a few things users are normally not aware of, like making sure that the random seed is frozen for a run, bugs in programs are caught (as long as needed) and that omissions in the `texmf.cnf` settings are compensated for. In addition TEXexec provides a few features for combining and manipulating PDF files.

The latest versions of TEXexec also support so-called ctx files. These are files in XML format that describe a process, additional pre- and postprocessing needed, styles and modules to be used, etc.[3] This means that one can easily configure projects without the need to tweak source files or editor setups or give explicit commands. Think of situations where an XML file (or bunch of files) has to be converted to another variant in order to be processed. TEXexec will only do that conversion when needed. In Figure 1 we show the file that is used in the MathAdore project.[4] The source file contains OpenMath and what we call 'shortcut math' and after normalizing this to OpenMath (first conversion) we convert the math to content MathML (second conversion).

The source file contains a reference to this ctx file and when TEXexec is applied to the source file, it will take the appropriate actions. Such a reference looks like:

```
<?ctx-dir job ctxfile ../mathadore.ctx ?>
```

Here "`ctx-dir`" denotes a ConTEXt directive.

When dealing with a TEX file, TEXexec will scan the first line for comments that serve a similar purpose.

Handling the utility file

For a long time TEXutil was called from within TEXexec to handle the utility file that collects the index entries, tables of contents, references, etc. Nowadays this functionality is integrated in TEXexec which is more efficient. We also took the opportunity to enhance the sorting features so that one can mix language specific sorting rules.

The original TEXutil is also responsible for some other manipulations, like analyzing graphics. That kind of functionality has been moved to other scripts and more modern ways of dealing with such issues. Because we were in a transition stage to Ruby scripting, it was a good moment to say goodbye to TEXutil and concentrate on building a more extensive set of tools.

[3] Although one can use the `ctx` suffix for ConTEXt related TEX files, this is normally a bad idea.

[4] This project will provide highly interactive math to schools and is conducted in cooperation with the University of Eindhoven.

```
<?xml version='1.0' standalone='yes'?>
<ctx:job>
    <ctx:message>mathadore</ctx:message>
    <ctx:preprocess suffix='prep'>
        <ctx:processors>
            <ctx:processor name='openmath' suffix='om'>texmfstart
                --direct xsltproc
                --output <ctx:value name='new'/>
                kpse:x-sm2om.xsl <ctx:value name='old'/>
            </ctx:processor>
            <ctx:processor name='mathadore' suffix='prep'>texmfstart
                --direct xsltproc
                --output <ctx:value name='new'/>
                kpse:x-openmath.xsl
                <ctx:value name='old'/>.om
            </ctx:processor>
        </ctx:processors>
        <ctx:files>
            <ctx:file processor='openmath,mathadore'>v*.xml</ctx:file>
            <ctx:file processor='openmath,mathadore'>h*.xml</ctx:file>
            <ctx:file processor='openmath,mathadore'>openmath*.xml</ctx:file>
        </ctx:files>
    </ctx:preprocess>
    <ctx:process>
        <ctx:resources>
            <ctx:environment>o-m4all.tex</ctx:environment>
        </ctx:resources>
    </ctx:process>
    <ctx:postprocess>
    </ctx:postprocess>
</ctx:job>
```

Figure 1: A ctx file used in the MathAdore project

The tools collection

Instead of expanding TEXutil, we decided to spread functionality over multiple scripts. These can be recognized by their name: they all end with `tools`. If you call them using TEXMFstart there is not much opportunity for conflicts with existing tools.

Each tool comes with a manual, so we will not discuss details here.

ctxtools

This tool provides ConTEXt related features, like generating generic pattern files (so that we are independent), providing editor syntax checking files derived from the generic ConTEXt interface definition (handy for lexers), generating documentation (from the ConTEXt source code), updating ConTEXt (by downloading an archive and regenerating formats), etc.

rlxtools

The 'r' represents resources, normally graphics, the 'l' stands for libraries, and the 'x' (indeed) for XML. This tool can analyze graphic files and manipulate resources using other programs. For instance it can be used to downsample files at runtime, to handle special color conversion, and to convert graphics to formats acceptable for TEX. By using the runtime converters one can build workflows without the need to rely on additional scripting. There is a dedicated manual on this topic so we will not bore you here with yet another blob of XML.

xmltools

You can use this tool to do a simple analysis on an XML file. Another option is to generate a directory listing in XML format. In both cases, the result can be fed into ConTEXt and used in the process. A more obscure option is to generate images from MathML snippets. This script will without doubt include more features in the future.

pdftools
This is work in progress. One can for instance roughly analyze PDF files. It also provides a way to manipulate colors in PDF images but that feature is now supported in ConTEXt directly.

textools
Users will seldom need this tool. It can fix things in a TDS compliant tree (for instance when the standard has changed), it deals with a few cross platform issues, it can help you to create so-called TPM archives (and is meant for ConTEXt module writers) and it can merge updates into your tree.

mpstools
In the future this tool will host the now standalone MetaPost to PDF wrapper (mptopdf) as well as the cropper (both are still Perl scripts).

tmftools
This script encapsulates some of the functionality of the Ruby based kpsewhich functionality that we use. In the future we may completely move away from the binary because the script is just as fast or faster when it serializes the database. The script can act as a kpsewhich server. The script can also analyze the tree for duplicates.

runtools
Because TEX is multiplatform and because we (need to) run services on multiple platforms, we use this script to do things normally done at the console (shell). It just loads the given Ruby scripts with the appropriate library. We also use this tool to generate the ConTEXt distribution.

exatools
This is a more obscure tool. It provides some features related to form based style control and web driven TEX processing that we use in projects.

pstopdf
This last one is not a collection like the previous tools. It started long ago as a wrapper for Ghostscript. It still provides this function and over the years we've added quite a bit of filtering to it (we just filter the things that Ghostscript fails on or gets confused from). In the meantime we cheat on the name since it also manages the conversion of bitmap images, especially cached downsampling, using ImageMagick as well as conversion from SVG to PDF using Inkscape.

texfont
This script has been around for a while now and is used to install (commercial) fonts. It generates metric files, map files, and a demo file so that one can see if things went right. ConTEXt does not depend on (ever changing) map file methods and loads map files on demand. You can generate map files for dvipdfmx with the previously mentioned `ctxtools`.

More

There are a few more scripts, like `concheck` (simple syntax checking) and `texsync` (synchronizing minimal distributions) but we will not discuss them here.

Integration

When setting up multiple TEX trees, the trick is in isolating them as much as possible. Because one can never be sure how distributions set things up, we revert to setting environment variables, which will then take precedence over the settings in a regular `texmf.cnf` file. In the TEXMFstart manual you can find more details on how we take care of this; here we only show an example of such an file in Figure 2.

When the tree flag is given, TEXMFstart will read this file and set the environment variables accordingly before it launches the program it is supposed to start. In fact, a tree specification can specify a file, but by default the `setyptex` one is taken.

```
texmfstart \
  --tree=f:/minimal/tex/setuptex.tmf \
  texexec test.tex
```

Since TEXMFstart can load several such files, we can also use this method to preset more environment variables, for instance pointers to resources like graphics. This is what the `-env` or `-environment` option is for, as in:

```
texmfstart --tree=f:/minimal/tex \
  --env=xyz.tmf texexec test.tex
```

The advantage of this variable setting game is that instead of cooking up scripts with statements like:

```
thread.new do
 ENV["something"] = "nothing"
 a = "texmfstart --tree=f:/minimal/tex --"
 system(a+"env=xyz.tmf texexec test.tex")
end
```

we can put the variable definition in a file and say:

```
# file   : setuptex.tmf (the less generic version have suffixes like cmd, sh, csh etc)
# author : Hans Hagen - PRAGMA ADE - Hasselt NL - www.pragma-ade.com
# usage  : texmfstart --tree=f:/minimal/tex ...
#
# this assumes that calling script sets TEXPATH without a trailing
# slash; %VARNAME% expands to the environment variable, $VARNAME
# is left untouched; we also assume that TEXOS is set.

TEXMFMAIN    = %TEXPATH%/texmf
TEXMFLOCAL   = %TEXPATH%/texmf-local
TEXMFFONTS   = %TEXPATH%/texmf-fonts
TEXMFPROJECT = %TEXPATH%/texmf-project
VARTEXMF     = %TMP%/texmf-var
HOMETEXMF    =

TEXMFOS      = %TEXPATH%/%TEXOS%

TEXMFCNF     = %TEXPATH%/texmf{-local,}/web2c
TEXMF        = {$TEXMFOS,$TEXMFPROJECT,$TEXMFFONTS,$TEXMFLOCAL,!!$TEXMFMAIN}
TEXMFDBS     = $TEXMF

TEXFORMATS   = %TEXMFOS%/web2c/{$engine,}
MPMEMS       = %TEXFORMATS%
TEXPOOL      = %TEXFORMATS%
MPPOOL       = %TEXPOOL%

PATH         > %TEXMFOS%/bin
PATH         > %TEXMFLOCAL%/scripts/perl/context
PATH         > %TEXMFLOCAL%/scripts/ruby/context

TEXINPUTS    =
MPINPUTS     =
MFINPUTS     =
```

Figure 2: Example `texmf.cnf` file

```
thread.new do
 a = "texmfstart --tree=f:/minimal/tex --"
 system(a+"env=xyz.tmf texexec test.tex")
end
```

This has not only a big advantage in terms of isolation (and maintenance) but is also more robust since one can never be sure if another thread is not setting the same variable too, thereby creating much confusion for all the other threads that use the same variable. Since TEXMFstart runs as a separate process, it can set its variables independently.

Whenever (on the ConTEXt mailing list) you see mentioning of something named `setuptex`, you can be sure that it relates to initializing a TEX tree (probably a minimal ConTEXt tree) in an isolated way.

Conclusion

In this short article we have tried to give you an impression of what is needed in order to make TEX usable in a diversity of today's environments. It was not our intention to be complete, because for that purpose we have manuals. One thing should be made clear: although TEX itself is pretty stable, the same cannot be said for the environment that it is used in. Just telling TEX to process a file is not enough nowadays. This also means that ConTEXt and its tools, in order to keep up, need to be adapted to current needs. On the other hand, by organizing the functionality in tools, and by using a modern and reliable scripting language like Ruby, users don't pay a high price for this. Most nasty details can be hidden from them.

KöMaL CD — The execution

ILDIKÓ MIKLÓS
`miklosildiko (at) komal dot hu`

What is KöMaL?

The first edition of Mathematical and Physical Journal for Secondary Schools (Középiskolai Matematikai és Fizikai Lapok — KöMaL [1]) appeared in the 19th century, on the 1st of January, 1894, under the name High School Mathematics Journal (Középiskolai Matematikai Lapok). Dániel Arany, a teacher of mathematics, founded the Journal. Since 1894, generations of mathematicians and scientists have developed their problem-solving skills through KöMaL. The best solutions with the names of the 14–18 year-old authors are printed in the periodical. KöMaL regularly reports on national and international competitions, prints articles on interesting results in mathematics and physics, and includes book reviews. At the present time, KöMaL is the oldest existing journal in Hungary.

From the beginning there have been competitions based on collecting points in the Journal, first only in mathematics and only in Hungarian and later in physics (from 1925) and in information technology (from 1981), and in several other languages: mainly in English (from 1965), but also in German, French, Russian, and Esperanto. For more than 30 years all the new problems have appeared in English as well as in Hungarian in the Journal. This means thousands of mathematics and physics problems and exercises have been published in English.

The periodical KöMaL is published in Hungarian nine times a year, and in English twice a year. Each issue is 64 pages.

The first CD

From the beginning until 1991 KöMaL was typed by hand. In 1995, we scanned all the pages of the issues and made a CD with the scanned pages. It is also viewable on the Internet, on the home pages of Educational Ministry, `www.sulinet.hu/komal`. We made a foundation of a searchable database for articles, problems, competitions etc. Of course, the scanned pages had limitations, such as problems with text quality, users being unable to search for words, copy problems, etc.

In 2004 we decided to type all 39,000 pages, and convert them to MathML. Then we made a CD which contains the pages of the Journal from 1994 to 2003. During the process we have faced several problems: from 1994 to 2001 the Journal was typed in plain TeX, with different macros. From 2002 we have been using LaTeX, with more and more packages. Finally Géza Makay, a teacher at Szeged University, made a converter from LaTeX to MathML, so now the CD has a web interface. He has also created a more user friendly database, which contains the title of the articles, the label and the mathematical type of the problems, the name of the authors and who made the best solutions of the problems, the final results of KöMaL competitions with photos of the overall winners, etc.

The project

To type the hand-made pages of the past years we needed manpower for the project. Finally in 2005 we were awarded a grant from the European Union and the nation of Hungary. So we could start the job with 24 low educated, disabled people. Some of these people have never touched a computer before they started this job. We had only two weeks to teach them how to use a computer and one more week to use LaTeX. So we had to create a curriculum which was flexible, very easy to learn and follow, and user friendly. After this short course the workers were given a computer, and they were expected to work in their own home. We found several free sources for learning LaTeX on the Internet, these were Oetiker–Partl–Hyna–Schlegl's 'LaTeX 2_ε in 78 minutes' and Gábor Csárdi's 'LaTeX 2_ε in 69 minutes'.

After the course they needed constant tutorial help, which was done by establishing an Internet connection. They also helped themselves using the Internet. The first questions were very simple but after a few months they could handle more and more complicated equations and tables in LaTeX.

Finalizing of the database

A couple points about the converter: e.g., we use equation numbering only with the `\tag` command (not with `\label`). We do not use `\textwidth` and `\textheight`, because of the web interface.

The workers load the database with LaTeX files. The database translates the LaTeX source with a fixed preamble and they can view the DVI results. Another person checks and corrects the LaTeX source. Over one year they were able to type and correct more than 15 years of the Journal.

The CD: "Aim at the Nobel prize"

This CD provides a unique opportunity to improve the knowledge of mathematics, physics and information technology of talented students, parents as well as teachers. The content is searchable according to multiple criteria, and one can even create his/her own worksheet using the selected problems or articles.

There were several famous Hungarian mathematicians who read the Journal and solved the problems, e.g., Paul Erdős and László Lovász, both of whom won the Wolf Prize (in 1983 and 1999, respectively). The CD contains photos of them at a young age.

We hope that having read the problem in English, one will be able to reconstruct its solution from the Hungarian text. Translating the whole KöMaL in English would probably be too big a task for us now, but we are considering it for the future.

For the next few years, we are planning further developments in the KöMaL archives, based on its current database: see `www.komal.hu/cd`. Our goal is to fill the database with the more than one hundred years of material, and also to be available in English.

References

[1] KöMaL: Középiskolai Matematikai és Fizikai Lapok, `http://komal.hu`.

A pdfLaTeX-based automated journal production system

THIERRY BOUCHE
Cellule MathDoc
UMS 5638 (Université Joseph Fourier & CNRS)
100, rue des Maths
Domaine Universitaire
38402 St-Martin-d'Hères, France
thierry dot bouche (at) ujf dash grenoble dot fr
http://www.cedram.org/

Abstract

We present the recent development of a production system for mathematical serials with both an electronic and paper version. The challenges were many: (i) no house style layout should be imposed, as the journals come from different publishing houses and may have very different typographical options; (ii) produce screen-optimised and printer-friendly output at once; (iii) avoid any duplication of information so that all aspects of the publications are always in sync (Web site metadata, table of contents…), thus (iv) generate on the fly article's page numbers, XML metadata at the published volume level from one master LaTeX source file tree. Using available technology (pdflatex, pdfpages.sty and `\write18`*), the proposed solution to these problems appeared amazingly simple and easy to use. However, we'll show that there is quite some room left for improvement.*

Introduction

At the end of fall 2003, discussions began in the French mathematical community about a consolidated effort for high-quality online publishing of our academically (meaning: independent and learned society) published research journals. One driving force of this project was the achievements of the NUMDAM digitisation program, which has more or less settled standards for delivery and navigation of a significant part of the mathematical literature (`http://www.numdam.org`).

Among the prominent features of NUMDAM, we have the rich set of metadata for each article, including tagged bibliographies, and the powerful search engine associated with it, written by Claude Goutorbe of Cellule MathDoc. Thanks to various matching tools provided by the AMS or developed internally, whatever sensible link can be provided is added to the Web interface, which is something our users very much appreciate.

It appeared after some investigations that what was almost straightforward to achieve in a retrodigitisation process could become a nightmare to produce in a natively digital environment:

1. Although all journals under consideration were produced with some flavour of TeX, each had a specific format with a primarily paper-only approach to the publication process.
2. Although all of them had a Web site, none of them had reliable processes to control whether the metadata exposed on this site was consistent with the reality of the paper issues.
3. Although bibliographies are such a routine object in the learned publication business, we could count over 20 ways of "structuring" them in the TeX files.

It turned out that, although we're now in the 21st century, the rather quaint copy-paste operation (a typical late 20th century hobby) was the main procedure on which all these journals relied for the most typical aspect of serials publishing: exposing the same data in many formats and contexts. Let's think for a moment about the starting page number of an article, which is a rather critical datum if you hope to find it somewhere. It will be printed within the article itself (where it is determined only at the last step of production, as it depends on the lengths of all the same volume's articles before it), probably an inner table of contents, possibly a back cover table of contents, presumably a Web table of contents, not to mention an eventual annual index, or third party uses of the data, as current contents or indexing databases services, that could nowadays be fed through OAI-PMH or RSS feeds…

For instance, let me give the following (anonymous) example: a respected journal once published a paper which, for some obscure (possibly scientific!) reason, was ultimately shortened by a paragraph or two in the proof reviewing stage. It was the first paper of the first issue of its year of publication in that journal, and was paginated 1–27 (hard coded in the TEX source) although its final form had 26 pages. The next article was thus paginated starting at p. 29 but the printer saved the 2 white pages. Thus, all the 2000 printed page numbers in this volume are wrong except the first 26.

Last minute changes and copy-paste are the two devils in journal production. A more obvious example: an accepted paper happens to have a serious scientific failure which is found after all the publishing process has been done. The author informs the journal in a hurry that they have to cancel it, of course they do. Now, all pages numbers are wrong for the following articles, go figure where they have already been disseminated!

In a retrodigitisation process, these issues are just annoying, but all you aim at is to create accurate and structured metadata describing an existing paper collection. Moreover, as it is a batch process on a large amount of similar data, high quality can be achieved at a reasonable cost. Production cost and complexity is an issue for our small journals, which heavily rely on voluntary effort by researchers in their spare time (as well as *Cahiers GUTenberg* which will enter the scene later on).

Good solutions to complex problems are simple

So. How do you produce a journal in such a way that you have detailed, accurate metadata in a versatile format, a powerful Web site with screen optimised versions of the articles, and yet the same paper version?

After some time spent in reviewing the existing more or less full answers to this question, mostly based on scripts, heuristics, external programs or auxiliary files, I happened to find one so simple that I think it deserves to be detailed here. In fact, it is so simple that I feel a little embarrassed to show the main steps in the small `\includearticle` macro on p. 48 while I spend the rest of the text discussing the troubles. At the time of this writing, 11 issues from 3 journals (including the latest *Cahiers GUTenberg*) were made using this tool.

This solution has been made possible rather recently thanks to the collaborative effort of many talented developers, and although I could achieve this because of the power of TEX macro language, I must confess that I never use TEX itself, but engines that understand an extension of TEX's primitives, yet have a full macro interpreter onboard, namely: pdfTEX with `\write18` enabled and (soon) Tralics.

Here is a short description of the system.

Principles

1. Any metadata is input at most once in the system, preferably in the relevant file.

2. Anything that is not deterministically determined by a given file, should stay away from that file.

3. Anything that can be computed, *should* be computed!

4. Do not reinvent the wheel, do not invent exotic formats that no one will master, stay pragmatic but avoid bottle-necks that would impact versatility of future use or quality of the output.

Implications

1. A journal is a set of volumes, made of issues, made of articles, plus various other material, mostly constant. The journal description belongs to the journal file, the volume description to the volume file, etc. Notice that a page number is essentially a by-product of a completed issue; except for the first page of an issue, it should be set nowhere.

2. As it is the de facto standard of math writing, AMSLATEX was chosen as the input format, with the minmal set of extensions as required by the subsequent processing. BibTEX was chosen for the bibliographies.

User interface

Of the relevant parts of a journal, I didn't implement the volume level (this would save copying *one* number) but tried to define an *issue*.

Claim 1 *An issue is entirely determined by*

- *its bibliographic data (journal, year, month, volume, issue),*
- *its first page number,*
- *the ordered list of the articles,*
- *and optional additional material such as advertising, disclaimers, obituaries…*

I write this (and only this) in the issue file:

```
\documentclass[francais,CG,Volume,
                         Couverture]{cedram}
\IssueInfo{}{46-47}{avril}{2006}
\SetFirstPage{1}
\SpecialNo{Les fontes (Brest 2003)}
\begin{document}
\makefront
   \includearticle{edito}
   \includearticle{atanasiu}
  [...]
   \includearticle{devroye}
   \includepub{pubyannis}
\makeback
\end{document}
```

The `\makefront` command makes the front matter of the paper volume (including the table of contents), `\includearticle` includes the corresponding article, `\makeback` makes the back matter, etc.

Each article obeys an AMSLATEX structure:

```
\documentclass[CG,francais]{cedram}
\usepackage{x,y}
\title[Formatons les formats de fonte]
       {Formatons\\ les formats de fonte !}
\author{\firstname{Luc} \lastname{Devroye}}
\address{McGill University,\\
etc.}
\thanks{L'auteur...}
```

Assuming that all the articles and other material are in final form (which means that they are in a directory of their own, and that an error-free source master file compiled with pdflatex has been compiled successfully with all cross-references resolved), when we compile (twice) the issue file, it will produce one big PDF comprising all inner pages of the paper volume; this is sent to the printer. It will also produce the pages of the cover, and an XML file with all the metadata for this volume. In fact, as a side effect, you'll also find in each article subdirectory a hyperlinked PDF with a first page added, so that everything is ready at once for shipping both the paper and electronic editions of that issue.

Architecture

LATEX is a "document preparation system"; it operates at the document level. I'm not convinced by systems that address the abovementioned issues by considering articles as subdocuments of a master document: they require a high level of normalisation of the sources to avoid conflicts (different macros with same names, cross references, etc.), many redefinitions of standard user commands which is very risky since users like shortcuts, and would yield broken links when you provide an article on its own. We can't expect that mathematicians will obey such strict rules, nor TEXnicians! Thus the relevant document unit in a journal is an article, preferably isolated in a specific directory in order to avoid conflicts with input of figure names, etc. It should be compiled individually and produce a nicely hyperlinked and searchable vector PDF. The metadata relevant to the article are standard: authors' data, title, abstract, keywords, subject classification, dates, bibliography. The volume, issue, page numbers are not part of the article itself, as it can be moved at any time without affecting its scientific content, thus without edits. Of course, an article is prepared for a journal, so that info should be present in the article file, and determine the layout and many typographical options.

Starting from the obvious observation that nobody but TEX knows how long an article is, when considering its source, I eventually understood that the only reliable solution for setting error-free page numbers was to ask TEX to do so. Of course, you could compile a volume with a perl script that would compute things, compile articles, examine the produced PDF to deduce page numbers, modify the articles, recompile, etc. These are heuristics, and will be broken at the first discrepancy between the paper volume and the model volume assumed by the script. In some sense, as long as articles have a "bibliographical" reference, we're still in the retrodigitisation paradigm when producing an electronic edition: it is the paper model that endows the article with its metadata, so it is by assembling the paper volume that we can deduce the required data to get the final articles. But, more generally, the same applies to purely electronic serial publications: even the table of contents of an incrementally growing online volume is something that is generated as the last step when the latest article is added. Only flat repositories like arXiv can bypass this question.

Implementation

Articles

As far as articles are concerned, the `cedram` class is simply `amsart`, with a few features added. Namely:

- some extra metadata fields (provision for bilinguism, journal dates, ...);
- the automatic inclusion of a configuration file at `\begin{document}`;
- a 'lastpage' trick;
- the facility to store the literal TEX code of a macro argument or an environment's contents and write it to auxiliary files in various formats (by overloading standard macros);
- hooks added in the presentation code so that all known layout options can be easily implemented;
- some ad hoc definitions for various theorem styles;
- a journal option to load the journal file defining all constant metadata and make-up for that journal;
- some more class options, mostly for compatibility (by default, the class requires hyperref, pdflatex, T1 encoding, Latin Modern fonts, ...).

There is a light version called 'special' for things that should look like an article but do not have its full features (editorial statements, e.g.).

When compiling an article at its final stage, it reads a possible configuration file that might override options and provide the needed metadata (issue info, first page), writes out the screen-optimised PDF (with a first page added, kind of an offprint cover, meant to identify more clearly the article origin when it will travel the net on its own), a `.cdrsom` file which contains a TEX command providing all the data pertaining to this article that could be used to generate the corresponding TOC line in whatever format, and a `.cdrxml` file that contains an XML-like snippet with all the metadata for this article.

Volumes

A volume is made using the same `cedram` class, with an option 'Volume', so that all the typographical options are the ones of the journal. There are some specific options to this mode of operation, such as 'Couverture' which will generate the cover, 'CouvTires' the covers for author's (paper) offprints ...

Let me explain what it does line by line, which will show how it works, and why it is so simple and reliable.

```
\documentclass[francais,CG,Volume,Couverture]{cedram}
```

This sets the volume mode for *Cahiers GUTenberg* (CG), with French hyphenation patterns for the editorial material surrounding articles, and will generate a cover.

```
\IssueInfo{}{46-47}{avril}{2006}
\SetFirstPage{1}
\SpecialNo{Les fontes (Brest 2003)}
```

These set variables: issue number (CG has no volumes), month and year of publication, starting page number of the first article, title of the issue when relevant. These variables are available during the whole LATEX run, as well as written to auxiliary files.

```
\begin{document}
\makefront
```

In article mode, many things happen at the point of `\begin{document}`—but not in volume mode, as far as I can tell! The `\makefront` call could have been automated here. In any case, this command sets `\pagestyle{empty}`, and inputs `CG-front.tex`, which in turn inputs the definition files for the front matter (title page, administrative data, summary). It also inputs a void file that can be populated locally for special occasion issues. The summary is a container constant source file making use of the issue variables and inputting a summary data file with some fixed name which is generated later on (thus the necessity of two runs to complete an issue). In fact, the summary is a 'special' item, thus a complete LATEX file which is compiled during the run, in a subprocess similar to the 'article' case below. The `\makefront` macro ejects all remaining material to be printed, goes to the next odd page, and sets the page counter of the master document to the value given by `\SetFirstPage`.

```
  \includearticle{devroye}
```

This is the main operation, but maybe the simplest one. It is so simple that I include its entire definition here:

```
\def\includearticle#1{%
  \IncludeArticle[2]{#1/}{#1}%
  \ifx\@empty\articlesXML
    \gdef\articlesXML{#1/#1.cdrxml}%
  \else
    \g@addto@macro\articlesXML{ #1/#1.cdrxml}%
  \fi
  \ifx\@empty\articlesSOM
    \gdef\articlesSOM{#1/#1.cdrsom}%
  \else
    \g@addto@macro\articlesSOM{ #1/#1.cdrsom}%
  \fi
```

As we can see, `\includearticle` is just a shorthand for the more general `\IncludeArticle` that assumes that the article's master TeX file resides in a subdirectory with the same basename. Moreover, it stores in a macro the list of `.cdrxml` and `.cdrsom` files that will be dealt with at the end of the run. Going back a few lines in `cedram.cls`, we have:

```
\def\IssueInfo#1#2#3#4{%
 \tkkv={\ScreenMode\issueinfo{#1}{#2}{#3}{#4}}%
 \issueinfo{#1}{#2}{#3}{#4}}
\let\articlesXML\@empty
\tkkp={\setpage}
\def\pdflatex{%
    pdflatex --shell -interaction=nonstopmode }
\newcommand\IncludeArticle[3][2]{%
 \cleararticlepage
 \immediate\write18{echo '\the\tkkv
            \the\tkkp{\thepage}' > #2#3.cfg}%
 \immediate\write18{cd #2 && \pdflatex  #3}%
 \ifcdr@redoBibtex
 \immediate\write18{cd #2 && bibtex #3}%
 \immediate\write18{cd #2 && \pdflatex  #3}%
 \immediate\write18{cd #2 && \pdflatex  #3}%
 \fi
 \immediate\write18{cd #2 && \pdflatex  #3}%
 \includepdf[pages={#1-},noautoscale]{#2#3.pdf}%
}
```

The main article operation is thus the following.

1. The last page is ejected and, depending on the journal style, we go to the next odd page before dealing with the article.
2. The issue info has been stored globally and is written to the auxiliary file for the article, together with the current page number (a traditional `\write` could have been used here as well).
3. Then, the article is recompiled; this will use the given information because it reads the just created `.cfg` file at `\begin{document}`. Optionally bibtex and further pdflatex calls are executed.
4. Finally, the newly generated PDF is included (except, of course, for its first page) in the master volume being produced.

The trick here is that we can trust the page counter of the master document: this is the actual paper volume to be printed! Thus we can reasonably be sure that the value of `\thepage` is the correct value for the first page of the next article, which will be included precisely at this page. And this will remain true next time as we input the final PDF of the article right away.

```
\includepub{pubyannis}
\makeback
\end{document}
```

These last lines show that we can add advertising, or anything else. The counterpart of `\makefront` is `\makeback`: it includes almost static pages (instructions to authors, subscription info, etc.). In fact, many things happen at `\end{document}`, which is the only place where everything is known about the issue in final form: an XML file is written by processing the master's and all articles' XML snippets, summary data is generated by concatenation of all articles' summary lines, and the cover is built by compiling the adequate template.

Metadata and format questions

As long as printer and screen (Web) PDF files are considered, the described system has proved to work quite effectively. But, when you wish to produce versatile metadata from LaTeX source, you can expect troubles. All typeset material is done by LaTeX, thus the abovementioned `.cdrsom` files are perfect, thanks to the possibility of redefining any necessary macro on the fly to have different views on the same data (for instance, one journal has three summaries in it: one in French, with corresponding abstracts, another in English, both at the beginning of the paper volume, and another one set differently on the back cover, where actual titles are used: this is why I store nine fields in the `.cdrsom` files).

Apart from pragmatic reasons discussed earlier, there is a fundamental reason to prefer TeX source as the master for all metadata: math authors write their papers with TeX, and validate their scientific content on the printed result, which is where last minute corrections happen. With a full XML process, outputting TeX code after automated transformations, the correction process would be much more difficult to control, and could yield cases where there is simply no way to obtain the desired physical representation of the article, which is at the present time still the only long-term reference for the scientific content of the paper.

The first version of the cedram tools assumed a lot of postprocessing of the 'pseudo' XML files output by LaTeX. We had the TeX code somewhat sanitised, textual material converted to UTF-8 with variable success, and math expressions exposed as GIFs on our Web interface, thanks to `latex2html`.

My first idea in this respect was to use the kind of trick that is exploited in hyperref to produce properly encoded PDF bookmarks in order to write Unicode files. I was not able to achieve this myself. I also had a look at TEX4Ht which sounds like a good conversion tool from TEX to XML or HTML+GIF as an alternate presentation format. I gave up because of the huge number of parameters and files necessary to understand before producing output only somewhat similar to my expectations.

I am currently experimenting with Tralics [1], which might be the killer application: instead of asking pdflatex to write out a pseudo XML snippet for each article, that will need further processing, it can easily write structured code tweaked for Tralics, where all the data is the literal unexpanded TEX string, leaving all the conversion process from TEX data to Tralics, which is very good at doing so.

It even parses BibTEX files, but might also easily be used to structure `thebibliography` environments! For example, here is an excerpt from the file generated by the compilation of our example article.

```
\begin{xmlelement}{article}
\xmlbibcite {b8}{8}
\xbox{pagedeb}{149}
\xbox{pagefin}{166}
 \begin{xmlelement}{auteur}
   \xbox{nomcomplet}{\firstname {Luc}
     \lastname {Devroye}}
   \xbox{prenom}{Luc}
   \xbox{nom}{Devroye}
{\killparcode\begin{xmlelement}{adresse}{McGill
      University,\\  etc.\end{xmlelement}}
   \end{xmlelement}
{\killparcode\begin{xmlattelement}[fr]{titre}%
 Formatons\\ les formats de
 fonte !\end{xmlelement}}
   \begin{biblio}
    \bibitem{b8}J.~\textsc{Andr\'e}
     \pointir « Ligatures \& informatique »,
     \emph{Cahiers GUTenberg}, \no22,
     p.~61--86, 1995. \end{biblio}
 \end{xmlelement}
```

After some minor configuration, thanks to the fact that Tralics rather deeply understands TEX macros, Tralics will generate a wonderful, valid, XML, with all text converted to Unicode, and math to MathML. This XML can be exploited at once on our Web sites.

The only remaining question is whether the world is ready for MathML. Recent tests show that the quality of the display of MathML expressions in current browsers has drastically increased: it is now similar to TEX in readability (which relies a lot on fine positioning of sub- or superscripts), and it considerably enhances accessibility to the math content on the Net. The only remaining difficulty is that, as Tralics is a full TEX interpreter, it cannot generate easily a mixed format sanitising the text strings to well-formed Unicode XML but keeps a verbatim copy of the math formulas in TEX, which might be the most practical for many of our users in the near future ...

```
<?xml version='1.0' encoding='iso-8859-1'?>
<article>
 <pagedeb>149</pagedeb>
 <pagefin>166</pagefin>
 <auteur>
  <nomcomplet>Luc Devroye</nomcomplet>
  <prenom>Luc</prenom>
  <nom>Devroye</nom>
  <adresse>McGill University,\\ etc.</adresse>
 </auteur>
 <titre xml:lang="fr">Formatons les formats
                        de fonte !</titre>
 <biblio type='flat'>
  <bib_entry crossref='cite:b8'>
   <reference>8</reference>
   <bibitemdata>J.&#xA0;<hi rend='sc'>André</hi>
     « Ligatures &amp; informatique »,
     <hi rend='it'>Cahiers GUTenberg</hi>,
     no&#xA0;22, p.~61&#x2013;86,
     1995.</bibitemdata>
  </bib_entry>
 </biblio>
</article>
```

References

[1] José Grimm, "Tralics, a LATEX to XML Translator", *TUGboat* 24:3 (2003), Proceedings of EuroTEX 2003, pp. 377–388.

Server side PDF generation based on LaTeX templates

ISTVÁN BENCZE, BALÁZS FARK, LÁSZLÓ HATALA, PÉTER JESZENSZKY
University of Debrecen
Faculty of Informatics
Egyetem t.
H-4032, Debrecen, Hungary
`jeszy (at) inf dot unideb dot hu`

Abstract

We present a web-based addressbook application that can generate customized PDF documents using LaTeX template documents. The application is hosted on a server and users can access its functions using a web browser. A working LaTeX system must be installed only on the server side. Each registered user can manage his or her own addressbook. They can upload LaTeX templates and can generate multiple PDF documents from a template. Templates are customized to each selected recipient, substituting the appropriate addressbook data element into them. An example application might be an invitation card or a letter that must be sent to different recipients. Moreover, users can create simple documents (e.g. letters) using builtin templates and a simple web-based document editor.

Introduction

The Portable Document Format (PDF) has become one of the most widely used electronic document formats for publishing documents on the Web. It has many advantages that made it very popular. Some of them are the following:

- It is an open standard.
- It is device and platform independent.
- It is suitable for both viewing and printing.
- It is a file format, not a programming language like PostScript. A PostScript file contains code that must be interpreted, whereas a PDF file is rather a description, that results in faster and computationally less expensive processing.
- PDF files are searchable.

The goal of this paper is to give an overview of the tools and techniques that can be used to generate PDF documents in Java applications.

The first section presents a brief overview of the family of PDF tools that are available in Java.

In the following section we present our solution that is based on LaTeX template documents and on access to an external LaTeX system.

The last section is devoted to our sample LaTeX template-driven web application that generates PDF documents.

Overview of creating PDF documents in Java applications using conventional tools

This section gives an overview of the widely used solutions for the dynamic creation of PDF documents in Java applications. These tools can be classified as:

- XSL-FO formatters,
- PDF class libraries,
- reporting tools.

In the following we restrict our attention to open source solutions.

XSL-FO formatters

XSL-FO is an XML vocabulary for document formatting, a TeX-like typesetting language that uses XML syntax. It is a part of XSL, a family of W3C standards for the transformation and formatting of XML documents.

Because of the verbosity of the syntax, XML documents using the XSL-FO vocabulary are not edited manually. In order to use XSL-FO one needs an XML document and an appropriate XSLT stylesheet to transform it into another XML document that uses the XSL-FO markup vocabulary. (The transformation is executed by an XSLT processor, which is commonly available in Java environments.)

Then the XSL-FO document is converted into a readable or printable format by a so-called formatter. The most widely used output format is PDF.

Although the current version of the XSL specification became a W3C recommendation in 2001, none of the existing XSL-FO software products (including commercial products) implements the full standard.

Apache FOP [3] is an open source formatter, implemented in Java, that is a partial implementation of the XSL specification. FOP provides a Java API to access all of its functionality, thus it can be embedded into Java applications without difficulty.

XSL-FO might be a good solution if your data is in XML. There are ready-to-use XSLT stylesheets for standard XML document formats, such as DocBook XML, to transform them into XSL-FO. Writing your own stylesheet is not an easy job. Although there are graphical authoring tools, a sound knowledge of XSLT and XSL-FO is required.

PDF class libraries

Several Java class libraries are available to create and work with PDF documents. Unfortunately most of them are commercial products.

For example, fourteen PDF class libraries are listed in the appropriate category of the Google Directory [1] at the present time, and only three of them are available as open source software. Another reference [2] provides a list of open source PDF libraries in Java and contains six entries at present.

PDF class libraries can be classified as low-level or high-level.

Low-level PDF libraries provide low-level access to the contents of PDF documents and allow the creation of PDF documents in Java applications. To work with these APIs the programmer must be quite familiar with the PDF document format. It might be very difficult and cumbersome to use them.

In contrast, high-level PDF libraries use object models to model the logical structure of PDF documents. These logical models consist of Java objects that represent building blocks such as pages, chapters and paragraphs. Manipulating the object model programmers can access and modify the content of the underlying PDF documents.

PJX

A typical example of a low-level PDF library is PJX [4]. In order to use it one must know all about the PDF document format.

PDFBox

PDFBox [5] is a high-level class library. According to the project's web site it is used in several open source and commercial software products.

It allows the programmer to access and manipulate individual pages within a document. The content of pages can be accessed as a stream of objects, and it is easy to add text and images.

Unfortunately the API does not provide access to higher level building blocks such as chapters or paragraphs. For example, in order to add some text one must position to the right location within a page.

Although the API documentation is quite good and there are also some example programs and a developer's guide, unfortunately the latter is not very extensive.

iText

iText [6] is another high level class library that is more user friendly than PDFBox. It uses a higher level abstraction of documents. The building blocks of documents are chapters, sections, paragraphs, list, tables etc. This model looks like a document object model of an XML document. It is well documented; a very good tutorial is also available. According to the project web site, a book on iText will be published by Manning Publications this year.

Reporting tools

These are software tools that can generate business reports based on templates and data in databases and other data sources. Visual report designers may assist in the preparation of the reports. Templates are typically stored as XML documents that can also be edited by hand.

For a comprehensive list of open source reporting tools see [7]. Reporting tools offer varying features and capabilities; for example, they support different data sources and output formats. Some of them can produce PDF output and some can not.

JasperReports

JasperReports [8] is an excellent and powerful open source reporting tool that is written entirely in Java. It has a Java API that provies full programmatic control over the entire reporting process form report definition to report generation.

Report templates are defined by XML documents or defined programmatically, but open source and commercial visual report designer tools are available too. Compiled templates can be populated with data that is passed as parameters by the application or that comes from various data sources. A wide range of data sources is supported, such as relational databases (via JDBC and also via Hibernate), EJBs, XML documents and CSV files. When a template is filled the resulting report can be viewed, printed or exported to PDF, XML, HTML, CSV, XLS or RTF.

JasperReport is a professional-quality tool with many other features, such as i18n and integrated charting support.

DataVision

DataVision is an open source reporting tool that is very similar to JasperReports. It is also open source and written in Java, and it can be incorporated into a Java application easily. Reports can be created using a visual report designer tool and stored as XML files that can also be edited manually. The generated reports can be viewed, printed and exported to tab or comma-separated text files, DocBook, HTML, PDF and XML.

Compared to JasperReports, it has fewer features, for example it supports only relational databases (via JDBC) and plain text files as data sources.

It is mentioned here because to the best of our knowledge it is the only reporting tool than can export to LaTeX. Note that it uses LaTeX only as an output format; the user may use the resulting LaTeX documents to produce PDF or PostScript files. DataVision itself does not interpret LaTeX files to produce PDF, it uses the iText PDF library instead to generate PDF files directly.

Problems with the above solutions when using LaTeX templates

Some problems with the solutions presented in the preceding section are summarized below.

Problems with XSL-FO

Lack of stylesheets for non-standard XML formats
If data is stored in a non-standard XML format and a stylesheet is not available to transform it into XSL-FO, it may be a difficult task to create an appropriate stylesheet.

Problems with PDF class libraries

Lack of flexibility As documents are created programmatically, any change in the output PDF file requires modification of the source code and the application must be recompiled.

Difficulty of use Low-level PDF class libraries require in-depth knowledge of the PDF format, making it extremely difficult to generate a PDF file. Even in the case of high-level libraries it may be difficult to achieve the right text layout.

Problems with report generators

Non-general purpose They are useful for generating business reports that contain tables and charts, based on data sources. They may not be the best solution to generate conventional documents such as letters. Typesetting large chunks of text and achieving the right layout may be difficult.

Common problems

Quality The aesthetic quality of the generated PDF documents is often poor compared with PDF files that are produced by LaTeX.

Java-TeX integration

Accessing TeX from Java

TeX and LaTeX offer the highest typographic quality. They can produce publication-quality PDF files with a professional appearance. It would be very useful if Java applications could benefit from it.

Unfortunately we have no knowledge of any existing standard tool to integrate Java and TeX.

Such an integration may work as follows. To produce high quality PDF output a Java application generates a TeX file. This is a trivial task since TeX source is plain text. Then the resulting TeX file is passed to a TeX system, that will turn it into DVI, PostScript or PDF.

The TeX system is accessed by using the `java.lang.Runtime` class, which allows the Java application to interface with the operating system. The application can have complete control over the TeX compilation process, it can interrupt the process if necessary and it also has access to the files that are produced by the TeX system.

Extending the above scenario with the use of TeX templates offers greater flexibility. In this case the Java application does not generate a TeX file from scratch, but it reads a template and populates it with data. (Report generators operate in this way.)

Template engines

As described in Wikipedia, a template engine is a piece of software that processes an input text (the template) to produce one or more output texts. Template engines are widely used and very popular in web application development to create dynamic web content. Their most important advantage is that they separate application logic from web page layout.

Many Java-based template engines are available; see [10] for a list of open source Java template engines. They are used not only on the server side to generate HTML, but also they may be used in other applications to produce arbitrary textual output, even source code.

Generating LaTeX sources using FreeMarker

FreeMarker [11] is one well-known general-purpose open source template engine implemented in Java. Although it is typically used to generate HTML web pages in servlet-based MVC applications, we use it to produce LaTeX sources based on templates, that are turned into PDF.

FreeMarker has a powerful template language. Directives such as `if`, `switch` and `list` provide programming capabilities, and other common programming language constructs as variables, expressions and user defined functions are also available in templates.

Just as in the case of web applications the template engine is frequently used to incorporate database content into templates. In the example below we use Hibernate to access a relational database.

Hibernate [12] is the most popular solution for object-relational mapping (ORM) in the Java world. Hibernate provides transparent persistence for Java objects, that allows applications to store, update and delete objects in a relational database. It also provides query and object retrieval facilities. It is simple to use FreeMarker and Hibernate together.

Here is an example template fragment for producing a LaTeX table with FreeMarker:

```
\begin{tabular}{ll}
\toprule
Title &   ISBN\\
\midrule
<#list HibernateUtil.query("from Book b
  where b.year = 2006 order by b.title")
    as book>
    ${book.title}   &   ${book.isbn}\\
  </list>
  \bottomrule
\end{tabular}
```

In the example, `HibernateUtil` is a helper class whose static `query(String query)` method executes a HQL query,[1] and returns query results as a list of objects. Here we retrieve all books in the database that are published this year sorted by title. The table contains titles and ISBN numbers of the books and the output would look like this:

Title	ISBN
Aglaja. Apokrif	9630779668
Kazár szótár	9637448306
Utazás a tizenhatos mélyére	9631425169

Related projects

Although they have not influenced our work, the NTS and $\varepsilon_{\mathcal{X}}$TeX projects must be mentioned here.

NTS stands for New Typesetting System. The goal of the project was to re-implement TeX in Java, but it was discontinued.

NTS has been replaced by $\varepsilon_{\mathcal{X}}$TeX [13], that is a TeX-compatible typesetting system written in Java. Originally it was started as an attempt to enhance NTS, but later the entire system was rewritten from scratch.

The system is under development. Although a downloadable installer is available at the website of the project, the development is currently in pre-alpha stage.[2]

The project is a very promising initiative, but there is much to do. If it becomes available it will provide a more flexible Java-TeX integration.

The TeX Catalogue [14] contains two packages that support database access, namely SQLTeX and LaTeXDB.

SQLTeX is a Perl script that reads an input TeX file containing SQL commands and produces an output in which the commands are replaced with the results. LaTeXDB is a similar preprocessor but it is

[1]HQL stands for Hibernate Query Language, the fully object-oriented query language of Hibernate.

[2]Namely, it is only a development release that is not "feature complete". The next so-called alpha release will be delivered for more general testing.

implemented in Python. Both packages support only MySQL databases.

In either case, TeX input files may contain constructs that look like commands (for example `\sqldb`, `\sqlrow` or `\texdbconnection`) but are not in fact TeX commands. (This means that TeX files containing them will not compile.) They will be interpreted and replaced by the preprocessor to produce TeX files that should compile without any errors. In that sense, SQLTeX and LaTeXDB operate in a similar way to FreeMarker, but using TeX syntax.

A sample web application generating PDF

We have developed a web application to demonstrate the above approach in practice. Using the web application requires registration. Each registered user can manage his or her own addressbook and can generate PDF files based on LaTeX templates. The user selects entries of the addressbook and these are used to populate the template with data. A separate PDF file is generated for each selected entry that will be offered for download in a single ZIP file.

For example, this can be used to generate letters in PDF that are customized for each recipient. The PDF files are produced by LaTeX, thus guaranteeing a certain quality. Many users do not have LaTeX installed on their computer, but via the web application they have access to a LaTeX system. (It is also possible not to use the addressbook at all, and simply produce single PDF files.)

After logging in users have the following options:

- manage addressbook (add, delete and modify entries),
- upload an existing template and generate PDF(s),
- create a new template with a simple web-based editor and generate PDFfile(s).

If the third option is selected the user is presented with a list of predefined templates. These templates are LaTeX document skeletons that are stored on the computer hosting the web application. The following templates are installed by default: article, book, report, letter, empty.[3] The document editor is initialized with the selected template.

The templates that are uploaded or edited by the user should be valid LaTeX documents that should compile without any errors, although they may contain constructs that have special meaning. Text surrounded by `'@'` characters is a variable reference, and a replacement text will be substituted for it.

Some variable references have a predefined meaning, for example

- `@current.name@` means the full name of a person in an addressbook entry;
- `@current.name.firstname@` is the first name of a person in an addressbook entry;
- `@current.addresses.country@` is the country of the default postal address of a person in an addressbook entry;
- `@current.addresses.home.zipcode@` means the zip code of the home address of a person in an addressbook entry;
- `@current.phonenumbers.office@` is the office telephone number of a person in an addressbook entry.

These variable references can be used to generate multiple PDF files from a single template based on addressbook entries. Any other variable references such as `@signature@` are called static variable references, which will be replaced by static replacement text.

The user is presented with a list that contains all static variable references that occur in the template. For each of them a replacement text may be specified.

The next step is to select the output format, the possible choices being DVI, PostScript and PDF.

If the template does not contain any variable references, or contains only static variable references, then a single result file will be generated. Otherwise as the last step the user must select at least one addressbook entry, and a DVI, PostScript or PDF file will be generated for each of them.

The results are offered for download in a ZIP file that contains the generated DVI, PostScript or PDF file(s) together with the log file(s) and LaTeX source(s).

The `\write18{`*command*`}` construct allows the execution of operating system commands and is a potential security risk. Thus, `\write18` should be disabled, especially in web applications such as this (normally this is the default in TeX systems).

The following technologies and software products were used in the development: JDK 5.0, Apache Tom-

[3]Additional templates can be added easily.

cat, JavaServer Pages (JSP), PostgreSQL, and Hibernate. Note that we did not use FreeMarker, as there was no need for such a complex template engine in this application.

References

[1] A list of PDF class libraries for Java. `http://www.google.com/Top/Computers/Programming/Languages/Java/Class_Libraries/Data_Formats/PDF/`

[2] Open source PDF libraries in Java. `http://java-source.net/open-source/pdf-libraries`

[3] Apache FOP. `http://xmlgraphics.apache.org/fop/`

[4] PJX. `http://www.etymon.com/epub.html`

[5] PDFBox — Java PDF library. `http://www.pdfbox.org/`

[6] iText, a free Java-PDF library. `http://www.lowagie.com/iText/`

[7] Open Source Charting & Reporting Tools in Java. `http://java-source.net/open-source/charting-and-reporting`

[8] JasperReports. `http://jasperreports.sourceforge.net/`

[9] DataVision. `http://datavision.sourceforge.net/`

[10] Open Source Template Engines in. Java `http://java-source.net/open-source/template-engines`

[11] FreeMarker. `http://freemarker.sourceforge.net/`

[12] Hibernate. `http://www.hibernate.org/`

[13] $\varepsilon_{\mathcal{X}}$TEX. `http://www.extex.org/`

[14] The TEX Catalogue Online. `http://texcatalogue.sarovar.org/`

Managing a math exercise database with LaTeX

PÉTER SZABÓ
Budapest University of Technology and Economics
Dept. of Computer Science and Information Theory
H-1117 Hungary, Budapest, Magyar tudósok körútja 2.
`pts (at) fazekas dot hu`

ANDRÁS HRASKÓ
Fazekas Mihály Fővárosi Gyakorló Ált. Isk. és Gimnázium
Horváth Mihály tér 8.
H-1082 Budapest, Hungary
`hraskoa (at) fazekas dot hu`

Abstract

TeX is a good tool for creating beautiful books, especially when the book contains a lot of math formulas. It is not rare that TeX is used to typeset a view of a database, by generating TeX source from the database text, possibly using XML as an intermediate format. Some TeX packages and formats support reading XML data directly.

In the matbook *project we have created a database of math exercises for special class secondary school students, as well as solutions and instructions for teachers. The data is organized in a tree structure of custom LaTeX environments in* `.tex` *source files. LaTeX reads these data files several times for generating the books. CVS is used for data replication and concurrent co-authoring. We are planning to switch to using a LaTeX-to-HTML translator to publish the database on the web.*

This paper presents the simple software architecture of the matbook *project and the design decisions we made concerning software and workflow, and it also compares* matbook *with other approaches such as big content management systems and TeX-enabled wikis.*

Non-standard use of TeX

The original purpose of TeX (and LaTeX) was typesetting beautiful books, journals and other printed material.

Novel uses include preparing slides for talks, developing software and its documentation together (e.g. web and ltxdoc), typesetting math formulas (e.g. Texvc [11]), typesetting printed and on-line HTML documentation together, rearranging PDF pages (pdfTeX with pdfpages.sty) and typesetting text generated from databases or other markup formats.

In the matbook project we use LaTeX to read a database of math exercises (in several passes), and typeset the material to books for students and teachers. This paper presents the software architecture and some implementation details of the matbook project, and it is also a case study of integrating excellent free software tools for low-budget publishing.

Project goals and products

The *Fazekas Mihály Secondary Grammar School* of Budapest [1] has been launching special mathematics classes for several decades, and is proud of its students winning national and international student competitions, and later becoming appreciated mathematicians. E.g. László Lovász, the well-known Hungarian mathematician, graduated from Fazekas in 1966.

Good mathematicians have good problem solving skills, and this skill can be best developed by solving problems and exercises. It is the responsibility of the teacher to choose the exercises for the students which best fit their learning curve. Talented students in a special math class need special attention. A lot of exercises and didactic experience have accumulated in Fazekas over the last few decades, and we have decided to publish this in printed form in Hungary; we are also planning to provide a web interface where all material is available. Thus matbook was born.

We are compiling a comprehensive exercise database (which also includes solutions, didactic advice, exercise lists for lessons and metadata for more accurate searching). Students and teachers in Fazekas are both working on extending this database, and we are developing software that would present this database to its audience. We are planning to publish exercise books (for students) and teachers' guides. If students buy the exercise books, teachers can give homework assignments from those books. (Of course, teachers will assign exercises whose solutions cannot be found in the exercise book.)

We are also planning to provide a web interface on which visitors can browse and view exercises, solutions etc., they can do a full text search, and they can also search for exercises in a given topic (specified using a set of predefined keywords). We already have a web interface for a comprehensive database of Hungarian secondary school math contest problems (which stores text in LaTeX format, and converts it to HTML using TTH [2]), and we'd like unify this with the matbook database.

Database structure

The database consists of

- *exercises* for the students;
- *hints* and *solutions* corresponding to the exercises, for the students;
- solutions for the teachers only;
- *remarks* and *didactic advice* corresponding to the exercises, for the teachers;
- a hierarchic taxonomy of *keywords* covering the fields of mathematics (e.g. prime numbers, trigonometry);
- association between exercises and keywords;
- organization of exercises into *chapters* and *volumes*;
- chapter and volume introduction text;
- ordered *exercise lists* prepared for obligatory and facultative lessons;
- *figures* referred to in the text, in EPS format.

Software components

- *volume typesetter:* a set of LaTeX macros to read the database in multiple passes, and typeset the book volumes;
- *indexer:* generates the keyword index at the end of the volumes (similar to *makeindex*);
- *web user interface:* with browse, view and search functionality;
- *consistence validator:* checks whether database files conform to the specifications.

Existing free software used: standard tools in a TeX distribution, the lmodern font family [3], GNU Ghostscript, sam2p [4], ImageMagick, CVS, Perl, the new magyar.ldf (part of [5]), husort.pl (Hungarian index processor, part of [5]), stuki.sty (structogram figure generator [6]).

We work in a Linux–Windows mixed environment, so it was our aim that all components except for the server part of the web user interface should run on both Unix and Win32. TeX tools we need are available on both systems. We decided to implement the indexer and the consistence validator as command-line Perl applications so it would be easy to port them across systems.

Database layout

We chose to store our data in structured text files rather than using a relational database, because it is easier to change the schema later, and we don't have to develop a custom user interface for data editing. XML is a good and widely supported structured text data model and syntax, but we prefer a format which is quick to type and easy to review for humans. YAML [7] is such a format. We finally chose the XML data model (for interoperability with other software), but a LaTeX-compatible syntax (for easy typing), which can be converted to XML without loss when needed.

As a master text markup format, we quickly rejected XHTML + CSS + MathML, mostly because it is tiresome to type a document in this format. Also, it is not possible to archive a rendered version of an XHTML text in a scalable way; it is not possible to specify typesetting hints (such as penalties); and MathML is not powerful enough: it is not possible to type the right, textual side of \cases in MathML; MathML still lacks some symbols. Moreover, with current browsers it is not possible to ensure acceptable visual quality: browsers render the same document differently, MathML support usually doesn't come out of the box, browser MathML fonts lack important symbols, browsers cannot hyphenate long words automatically, the visual output depends on the installed fonts and the browser window size (which the author of the text cannot control), browsers cannot break the line in the middle of a MathML formula etc.

We could have adopted a safe and easy to type markup format, similar to MediaWiki's WikiText format [8] or ŞäferTeX [9]. The MediaWiki software implements the text rendering engine of Wikipedia [10], and it lets authors insert math formulas in a subset of (AMS)LaTeX syntax. When the page is rendered, these formulas are interpreted and converted to images or MathML formulas by Texvc [11]. We have rejected MediaWiki because — as with XHTML — it doesn't give the author enough power to ensure perfect visual output quality. We did not use ŞäferTeX because its source code was not available, and it was not mature enough.

We could have invented our own markup format. Doing this would have required us not only to invent an excellent format, but to write a renderer (to both PDF and HTML), and document the format thoroughly, including tutorials and examples. This option was not feasible in our project.

Thus we have chosen a restricted subset of LaTeX as a markup format. Its advantages are: it has been available for a long time, the basic commmand set is well-documented, it gives the text author sufficient control over the visual quality of the output, and there are lot of fonts and packages we can use. We had to impose restrictions in order to keep our format convertible (primarily to XHTML + CSS + MathML). The most important restrictions on the document text are: it is forbidden to load packages or other files, define or change macros, use conditionals or other programming features, change catcodes, use the character `"` in the input (the "proper quotes" must be used), use conditionals, or insert figures with `\includegraphics` (we provide a more restricted command instead).

Once we settled on LaTeX as a text markup format, it was straightforward to use the same syntax for structuring the data, so that our database text files won't contain two alternative formats, and they can be syntax-highlighted or otherwise processed in text editors easily. However, plain LaTeX is not suitable for structuring. For example, it is not obvious to deduce where chapter "First" ends in this LaTeX source, without knowing the meaning and depth of `\section`:

```
\chapter{First}     \emph{First} content.
\section{Inside}    \emph{Inside} content.
\chapter{Second}\label{2nd} % dummy
                    \emph{Second} content.
```

The XHTML representation (using `<H1>` for chapter titles and `<H2>` for section titles) suffers from the same limitation.

Our data format solves the problem by specifying structure using custom LaTeX environments. The example above looks like this:

```
\begin{mchapter}{title={First}}
  \emph{First} content.
  \begin{msection}{title={Inside}}
    \emph{Inside} content.
  \end{msection}
\end{mchapter}
\begin{mchapter}{title={Second},id={2nd}}
  % dummy
  \emph{Second} content.
\end{mchapter}
```

When converted to XML, it becomes:

```
<mchapter title="First">
  \emph{First} content.
  <msection title="Inside">
    \emph{Inside} content.
  </msection>
</mchapter>
<mchapter title="Second" id="2nd">
  <!-- dummy -->
  \emph{Second} content.
</mchapter>
```

Please note that `\emph` is not converted, because it is part of the text markup, and not part of the structure.

Thus there is a simple mapping between XML and our data format:

- LaTeX environment with attributes (i.e. "key = value" pairs) ⟷ XML tag with attributes, properly escaped
- TeX comment ⟷ XML comment
- other LaTeX text ⟷ XML text (PCDATA)

This direct mapping makes it possible to use XML tools on our database. For example, we can use XSLT to do structural transformations on the XML, and we can use DTD or XML Schema validators to validate our database.

Database folders and files

The database is spread into several small text files in several folders. The files read each other (using `\input`). The points where the data must be split and the naming conventions for the files and folders are strictly regulated.

The database is replicated on each co-worker's machine, using the CVS [12] revision control system. People can work offline, and commit their changes back to the repository on the server a few times a day. Server failures and network connection slowdowns don't affect working hours seriously. CVS is smart enough to merge concurrent but independent changes of text files, and it enforces human interaction when a conflict occurs, so there is no danger of accidentally overwriting somebody else's changes. CVS also keeps old versions, so accidentally deleted text can be recovered any time later. (CVS merges files line-by-line, which complicates concurrent editing of binary files — but this limitation doesn't affect our project since we mostly use text files.) Subversion (= SVN [13]) is a newer and more advanced revision control system, and it won't be hard to migrate from CVS when those advanced features are needed. Both CVS and Subversion have clients on multiple platforms, including Unix and Win32. We use the standard `cvs` client on Linux, and TortoiseCVS on Windows.

The reason why the database is split into multiple files is that it is easier to transfer changes of smaller files in CVS, and it is also easier for humans to edit a few small files concurrently than to edit one large file. Usually multiple people are modifying the database at the same time, but most of the time they work in their own files, so no conflict occurs. Using multiple folders makes it easier to select the correct file for opening.

The file and folder layout also follows the LaTeX compilation process. Compilation always starts in the root folder (of the CVS tree). For example, to compile the first volume of Algebra, one runs "`latex volume_a_i`", which starts processing the file `volume_a_i.tex`. All other files belonging to this volume reside in the folder `chs_a_i` and its subfolders. Files `\input` are thus specified relative to the root folder, thus adding `../` is not necessary when referring to local `.sty` files. It is also convenient that all temporary and output files go to the root folder, thus subfolders are not changed during the compilation process.

LaTeX tricks

In this section we present some problems we faced when typesetting with LaTeX; solutions included.

Unified labels

This is a feature that makes it possible to refer to a `\label` defined in another volume. It is accomplished by reading `\newlabel` commands from the other `.aux` files, and adding them with both the label text and page numbers prefixed with the other volume name.

Volume split

Since teachers' guides can be several hundred pages long, it might be necessary to split them into multiple volumes. If this is so, the editor promotes some chapter boundaries to volume boundaries by adding the appropriate command to the source of the main `.tex` file. We chose the most portable ways to typeset these subvolumes: all subvolumes are separate LaTeX documents, which `\input` the main `.tex` file in a mode in which the `\shipout` of the unnecessary pages is cancelled.

The advantage of this method is that it doesn't require external tools (such as psselect), it works for both PostScript and PDF, and it can be run from a regular LaTeX IDE. Its disadvantage is its slowness. An alternative approach for PostScript would be generating and running a psselect command line for each volume. And for PDF, an alternative approach is selecting the appropriate pages from the main volume using pdfpages.sty.

Splitting the volume into real subvolumes (so that compiling a subvolume doesn't read the other subvolumes' LaTeX source) would not work, because it would render page numbers, the bibliography and inter-subvolume references incorrectly, or need additional programming.

Fuzzy keyword names

When specifying the list of keywords associated with an exercise, it is a big burden to specify a long keyword precisely (with spaces, punctuation etc.). In order to solve this, a Perl script generates keyword aliases, e.g. the first word of a keyword will become an alias for the keyword if this is not ambiguous.

String processing

Another idea in fuzzy keyword name matching is to match keywords after stripping spaces, punctuation, upper case and accents. This stripping had to be implemented in LaTeX, too. Since TeX doesn't have string processing primitives, we have to implement them using macros. Here is a mix of a macro definitions that shows the most important string processing tricks:

```
\def\stripit#1>{}\def\empty{}\def\space{ }
\def\rmonestar#1{\ifx#1\hfuzz\empty\else
  \if*\string#1\else#1\fi
  \expandafter\rmonestar\fi}
\begingroup\lccode`!` \lowercase{\endgroup
\def\oonespace#1 {\ifx\hfuzz#1\empty\else
  #1!\expandafter\oonespace\fi}}
\def\rmstars{%
  \afterassignment\rmstarsb\def\M}
\def\rmstarsb{%
  \edef\M{\expandafter\stripit\meaning\M
          \space\hfuzz\space}
  \edef\M{\expandafter\oonespace\M}
  \edef\M{\expandafter\rmonestar\M\hfuzz}}
```

The macro `\rmstars` above removes all stars (`*`) from a string. The string is given as an argument in braces, and the result — without the stars and all tokens having catcode 12 — is put into the macro `\M`. Example invocation: `\rmstars{a * B**cd} \show\M`

A rough outline of its operation: `\meaning` converts tokens to catcode 12, except for spaces, which are converted to catcode 10. Then `\oonespace` iterates over all spaces and converts them to catcode 12, too. Finally `\rmonestar` iterates over the tokens and removes all stars. Almost beautiful.

Read more about the primitives involved here in *The TEXbook* [14].

One or more solutions?

If there is one solution for an exercise, it should be prefixed with "Solution" (and not "Solution 1"). If there are more solutions, each of them should have a number, "Solution 1", "Solution 2" etc. By the time of emitting the 1st solution, we don't have the information whether there are more. How do we typeset it properly?

To solve problems like that, it is a common trick to use the `\label`–`\ref` mechanism. We emit `\label{exercise42-sol2}` at "Solution 2", and the next time the document is recompiled, at "Solution 1" we check for the presence of this label, e.g. with
`\@ifundefined{r@exercise42-sol2}{...}{...}`

Bibliography three times

When a `\cite` command with a new target is added to the document, it is necessary to run LaTeX three times: `latex doc; bibtex doc; latex doc; latex doc`. The 1st run of LaTeX records the `\citation` command to the `.aux` file. The BibTEX run generates the `.bbl` file. The 2nd run of LaTeX inserts the new `.bbl` file to the document, and it also records the `\bibcite` command to the `.aux` file indicating the new number to be displayed by the `\cite` command. The last, 3rd run of LaTeX makes `\cite` emit that number.

We speed this up by parsing the `.bbl` file at the beginning (before the first `\cite`), so the 3rd run of LaTeX is not necessary.

Conclusion and future work

The matbook project demonstrates not only the power, openness and flexibility of LaTeX, but is also an example of low-budget publishing using free software and a little scripting. matbook is also free software.

Our most important future goals are completing the exercise database and implementing the missing software components: a thorough consistence generator and the web user interface.

References

[1] Fazekas Mihály Secondary Grammar School of Budapest. `http://www.fazekas.hu/`

[2] TTH: the TEX to HTML translator. `http://hutchinson.belmont.ma.us/tth/`

[3] Bogusław Jackowski and Janusz M. Nowacki. *Latin Modern fonts: how less means more.* In proc. of EuroTEX 2005, pp. 172–178. `http://www.dante.de/dante/events/eurotex/papers/TUT09.pdf`, 2005.

[4] Péter Szabó. *Inserting figures into TEX documents.* In proceedings of EuroBachoTEX 2003.

[5] Péter Szabó. *Implementation tricks in the Hungarian Babel module.* In proc. of TUG 2004.

[6] Károly Lőrentey. *stuki.sty: Structograms in LaTeX.* `http://lorentey.hu/project/stuki.html.en`

[7] YAML: machine parsable data serialization format. `http://www.yaml.net/`

[8] WikiText: wiki markup language. `http://en.wikipedia.org/wiki/Wikitext`

[9] Frank Schäfer. *ŞäferTEX: Source Code Esthetics for Automated Typesetting.* In proc. of TUG 2004.

[10] Wikipedia: the free encyclopedia that anyone can edit. `http://en.wikipedia.org/`

[11] Texvc: TEX validator and converter. `http://en.wikipedia.org/wiki/Texvc`

[12] Karl Fogel and Moshe Bar. *Open Source Development with CVS.* 3rd Edition. O'Reilly, 2003.

[13] Ben Collins-Sussman et al. *Version Control with Subversion.* O'Reilly, 2004.

[14] Donald E. Knuth. *The TEXbook.* Addison-Wesley, 1984.

The making of a (TEX) font

TACO HOEKWATER
Bittext, Dordrecht
info (at) bittext dot nl

HANS HAGEN
Pragma ADE, Hasselt

Abstract

We want to introduce a new display font to the TEX community. The font is a digitization of a series of Duane Bibby drawings, commissioned by Pragma ADE. The digital version for use with ConTEXt is prepared by Bittext based on scans provided by Pragma ADE.

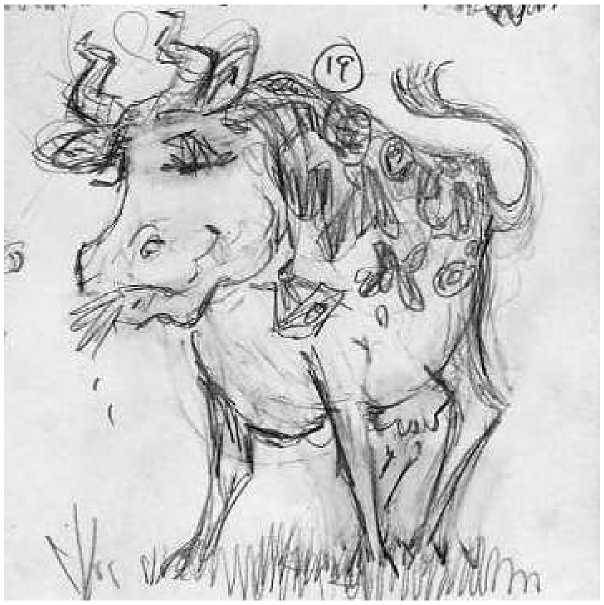


Figure 1: The first drawing

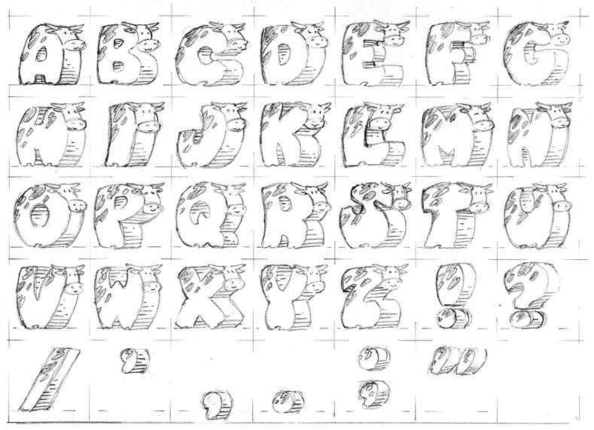


Figure 2: Rough design

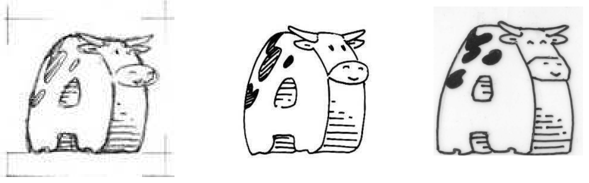


Figure 3: Shapes were drawn on a grid, and refined over time

Introduction

At TUG 2003 in Hawaii, Hans Hagen met with Duane Bibby. Hans was looking for some small images to enliven the ConTEXt manuals. A cutout of a very early sketch can be seen in Figure 1, but it was soon agreed that consecutive drawings were going to be an alphabet.

Nothing much happened after that initial meeting, until the beginning of this year when Hans picked up the thread and got Duane started drawing. The alphabet quickly progressed. Starting in a rather naturalistic style like Duane's 'normal' TEX drawings, but later progressing toward a much more cartoon-like style, as can be seen from the drawings in Figure 2.

For ease of use, it was clear that these drawings should ideally become a computer font. Taco Hoekwater agreed to take care of the digitization, and luckily the drawings were already prepared for that. As can be seen from the leftmost closeup in Figure 3, the cows are drawn inside a grid. This ensures that they are all the same size, which is a vital requirement for a font.

The center drawing in Figure 3 is a still rather roughly inked version of one of the in-between drawings (there were many). In this particular one you can see that the mouth of the cow was originally more or less oval, but in the final form (on the right) it became much more hexagonal.

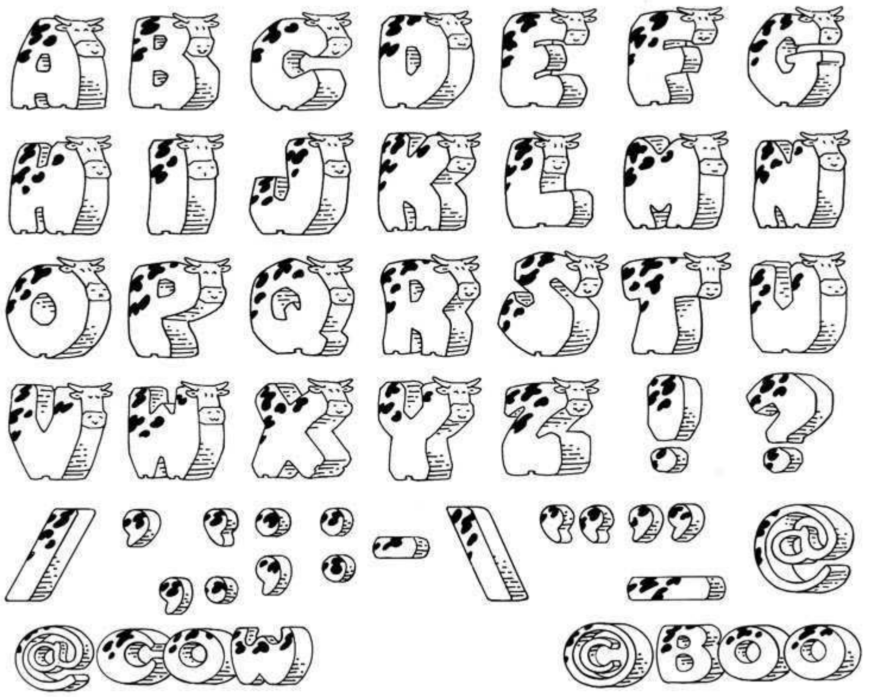


Figure 4: One of the original sheets, showing the alphabet and latin punctuation

Digitization

The original sheets were sent to Pragma ADE by regular mail in the beginning of March. Hans scanned the original sheets at 1200 dpi and then forwarded the images to Taco. There were four sheets in all, and one of them is shown in Figure 4. The other three contain a number of TEX-related logos and a few (mathematical) symbols.

Preparing the images

The first task in the preparation of the font was to create a set of bitmap images for use by FontForge's import command.

For this, the four sheets had to be cut up into many smaller pieces, each containing a single glyph for the font. This being intended as a decorative font, the character set does not even contain the complete ASCII range. Nevertheless, almost a hundred separate images were created.[1]

FontForge automatically scales imported images so that they are precisely one em unit high. After cutting the sheets up to pieces, the images therefore had to be adjusted so that they all had the same height. Without that, it would have been nearly impossible to get all the drawn lines in the glyphs the same width. Figure 5 shows the adjusted version.

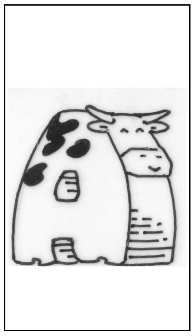


Figure 5: An imported bitmap images, with height adjusted

Automatic tracing

The autotracer in FontForge, which is actually the stand-alone autotrace program, does quite a good job of tracing the outlines. But, interestingly enough, only at a fairly low resolution. At higher resolutions it gets confused and inserts more than a quadratic amount of extra points for each resolution increase. Based on empirical tests, the images were scaled to 40% of their

[1]It would have been nice if this step could have been done solely with free software, but the Gimp turned out to be incapable of handling the 75 megabyte PNG images for each of the four scanned sheets. Adobe Photoshop was used instead.

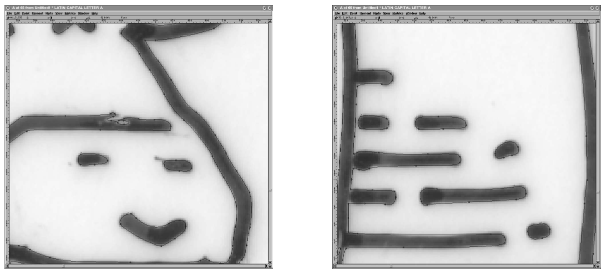


Figure 6: Close-ups of autotracer output

original scanned size, resulting in bitmaps that were precisely 1000 pixels high.

As was to be expected, the autotracer brought out many of the impurities in the original inked version, as you can see in the left image of Figure 6. Luckily, the number of places where manual corrections like this were needed was not so big to force us to reconsider the digitization process.

A more severe problem can be seen in the right-hand image of Figure 6. The drawings contain hardly any straight lines. For a font of this complexity, it turned out to be absolutely necessary to simplify the curves. Without simplification, the rendering speed in pdf browsers became unbearably slow. All of the near-horizontal stripes in the bellies were manually removed and replaced by absolute straights.

Hinting

The final stage in the font editor is to add the PostScript hinting. A screenshot of a manually hinted letter is shown in Figure 7.

This part of the work is in fact turned out to be one of the largest jobs, because it is necessary to find a balance between two possible extrema.

On the one hand, if there are no hints at all, that results in nice small fonts that render quickly, but poorly.

On the other hand, if there is a lot of hinting information, that creates a much better appearance but it slows down the rendering. And sometimes extra hinting produces worse rendering than less hinting, especially with non-commercial renderers.

A middle ground can be reached, but unfortunately only by doing all hinting manually, and that took quite a lot of time.

Finishing the font

The font was saved as two separate PostScript Type

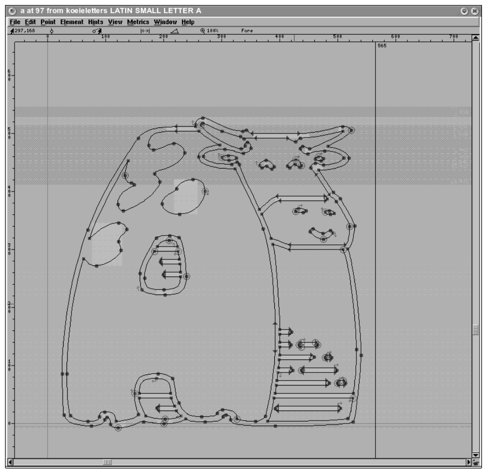

a at 97 from koeieletters LATIN SMALL LETTER A
File Edit Point Element Hints View Metrics Window Help

Figure 7: Finished outline, with hints

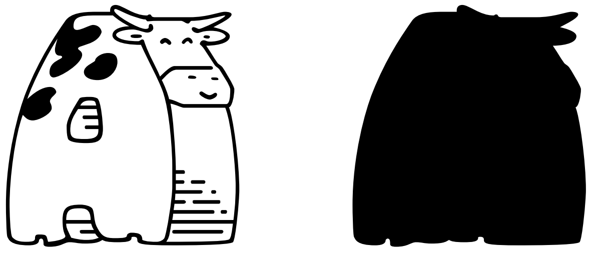


Figure 8: The final 'A'

1 fonts, one with the text glyphs and one containing the logo glyphs. The text font is named 'koeieletters', the logo font 'koeielogos'. 'Koeieletters' literally translates from Dutch to English as 'cowcharacters', but the word 'koeieletter' is also is used to indicate a really big character. Like in a billboard, for instance.

Eventually it turned out that we needed a second set of two fonts. Sometimes you want to have text in the cowfont but on top of a colored background. The background would then shine right through the hide of the cow and that was of course unacceptable. Hence, we also have the fonts 'koeieletters-contour' and 'koeielogos-contour'.

Figure 8 shows the final 'A', in the normal and the contour font.

Playing around

The original goal of this font was to enliven the Pragma ADE manuals. It would be a waste if we did not try to get the most out of the drawings provided by Duane, so we coded some rather silly effects in the font and its metrics. The final paragraphs of this article highlight a few of those.

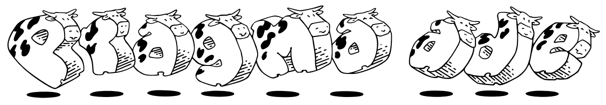


Figure 9: The Pragma ADE logo

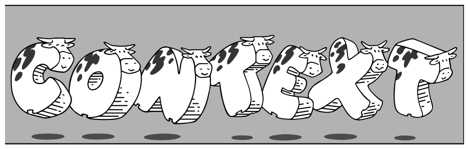


Figure 10: The ConTEXt logo

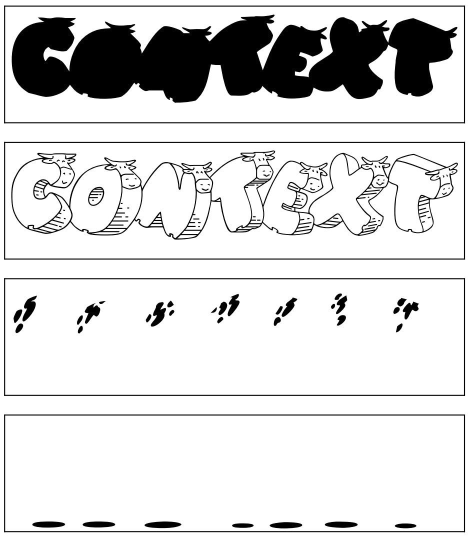


Figure 11: Four characters of the ConTEXt logo

Pragma ADE
The only lowercase symbols in the font are in the Pragma ADE logo itself, see Figure 9.

The ConTEXt logo
If you look closely at the ConTEXt logo in Figure 10, you can see that the shadows and the spots of the cows are drawn in different shades of grey (or colors). This is only possible because there are actually four characters involved instead of just the logo and the background contour, as shown in Figure 11.

Logo ligatures
The line on Figure 12 can be input directly as "`Cows in ConTeXt`", thanks to a handcrafted virtual font. This font contains a complete set of ligatures that travel from 'C at the beginning of a word', past 'e following the temporary ligature C_o_n_T', all the way to 't at the end of word'. Such word logos are defined for TEX, ConTEXt, and MP, see Figure 13.

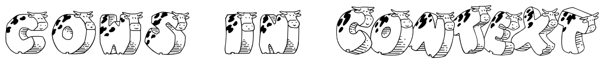


Figure 12: "Cows in ConTEXt" typeset

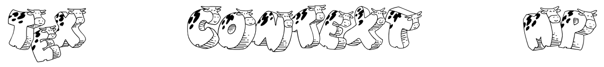


Figure 13: Word logos

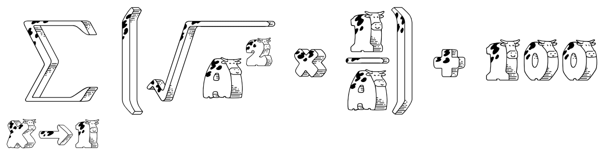


Figure 14: Cows in math mode

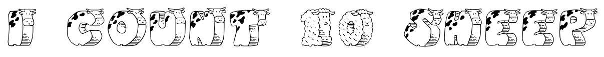


Figure 15: Old-style sheep

Mathematics
As is true for the collection of normal text glyphs, the math set is also not very extensive. But there are just enough math symbols to allow some example math formulas to be created. Virtual fonts make sure that the input is what you expect from TEX. See Figure 14.

Old-style Sheep
Seeing nothing but cows does tend to get boring after a while. To prevent the font from getting too predictable, we decided we needed some extra freshness. That is why the old-style numerals are actually sheep (see output in Figure 15):

```
    I count \oldstylenumerals{10} sheep
```

Final words

The koeieletter fonts can be downloaded from the Pragma ADE site. The fonts, needed typescript file and macros are part of the standard ConTEXt distribution.

Font installation the shallow way[*]

SIEP KROONENBERG
Rijksuniversiteit Groningen
Department of Economics
P.O. Box 800
9700 AV Groningen, The Netherlands
`siepo (at) cybercomm dot nl`

Abstract

For one-off projects, you can cut corners with font installation and end up with a more manageable set of files and a cleaner TEX installation. This article shows how and why.

Keywords

Font installation, afm2pl, afm2tfm, TrueType, pdftex, mapfiles

If you are putting together a flyer or invitation or book cover, then it would be nice if you could test a batch of fonts from your CorelDRAW or Illustrator CD, or your Windows font directory, without too much trouble and without polluting your TEX installation with a lot of stuff you are never going to use again.

This article takes you through the steps needed to use one or more fonts in one particular document. We won't really install the fonts; we just generate the files that TEX needs and leave them where TEX will find them, *i.e.* in the working directory. This makes it easy to take the project to another system, and easy to clean things up.

We will primarily use afm2pl to generate `.tfm` (TEX Font Metric) files. Later on, we show the steps required for afm2tfm. Both programs are simpler and much faster to use than the usual choice, fontinst. They create few intermediate or unnecessary files and do their job without virtual fonts. Virtual fonts and fontinst have their place, but sometimes there is no good reason to put up with the inevitable mess.

afm2tfm is available on all major free TEX implementations. afm2pl is part of current TEX Live distributions. Note that these programs are needed only to create the necessary font support files for TEX; once these files have been created, they can be used on any other system, whether or not it contains afm2pl or afm2tfm.

*This article appeared originally in slightly different form in MAPS 33, fall 2005.

An example

We use a decorative script font Pepita that Adobe bundles (or used to bundle) with some of its software.

pdftex needs the actual font file `epscr___.pfb`, its TEX font metrics file `epscr7t.tfm` and a mapfile containing an entry relating the two. First, we copy not only `epsrc___.pfb` but also `epsrc___.afm` to the working directory. We need the latter file to generate the `.tfm` file. Next, we enter the following commands on a command line:

```
afm2pl -p ot1 epscr___.afm epscr7t.pl
pltotf epscr7t
```

The extensions `.afm` and `.pl` are optional. The first command converts the `.afm` file to an (almost) human-readable text version of the desired `.tfm` file. The second command creates the more compact binary version.

Before we can use this font, we must tell LATEX about it. We do this with a font family definition file `ot1myfontfam.fd`:

```
\ProvidesFile{ot1myfontfam.fd}
\DeclareFontFamily{OT1}{myfontfam}{}
\DeclareFontShape{OT1}{myfontfam}{m}{n}{
  <-> epscr7t }{}
```

The prefix `ot1` indicates the encoding, which tells which characters occur at what positions. The next section will say more about encodings. The parameters to `\DeclareFontShape` are successively encoding, family name, weight (*e.g.* bold), shape, font file (without extension) and special options. You can normally leave this last parameter empty. With just one family member, we are not fussy about font characteristics and just pick defaults. We also leave this file in the working directory.

This is the code of our first testfile `exabasic.tex`, which uses this font:

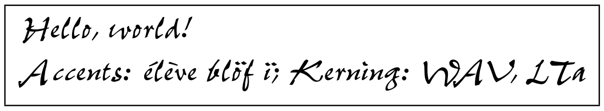
```
\documentclass{article}
\pagestyle{empty}
\pdfmapfile{=epscr7t.map}

\newcommand{\fancyfont}%
  {\fontfamily{myfontfam}\selectfont}

\begin{document}
\fancyfont
Hello, world!

Accents: \'el\`eve bl\"of \"i;
Kerning: WAV, LTa
\end{document}
```

The `\pdfmapfile` command causes pdflatex to read the file `epscr7t.map`, which tells pdftex how to get the font into the output file. The prepended '=' tells pdftex that it should read `epscr7t.map` *in addition to*, not instead of, the default mapfile, and that in case of a conflict `epscr7t.map` wins.

Now we are ready to compile `exabasic.tex`:

```
pdflatex exabasic
```

This is the result:



Hello, world!
Accents: élève blöf ï; Kerning: WAV, LTa

Encodings

We already mentioned encodings briefly. Now it is time to dig a little deeper, because it is a topic that can easily trip you up.

An encoding defines what character corresponds to which number. Only numbers between `0` and 255 are allowed. A `.tfm` file associates character metrics directly with character positions and doesn't know what position represents what character. TeX simply makes assumptions about this correspondence or encoding, and if you disagree with those assumptions then you need to load some macro package or other to tell TeX otherwise.

We hope that mainstream TeX will eventually move to Unicode, which is a comprehensive encoding of all conceivable characters, including far-eastern alphabets and mathematical symbols. When that happens, we can forget about encodings and also do away with many applications of virtual fonts. There are already some Unicode-based variants of TeX.[2]

For a PostScript `.pfb` or `.pfa` font, character metrics are stored in a separate `.afm` file. These metrics are associated with characters, not with character positions. Therefore you should specify an encoding to afm2pl or afm2tfm.[3] The same encoding must also be specified in the mapfile entry. A PostScript font usually has more characters than fit into a single TeX encoding.

A command-line option such as '`-p texnansi`' or '`-p texnansi.enc`' means that the encoding should be read from a file `texnansi.enc`. This encoding probably has a different internal name.

OT1 encoding

If you don't tell TeX otherwise, it assumes that you are using the OT1 encoding. This encoding uses only 128 of the 256 available slots. TeX creates missing accented characters from an unaccented base character and a separate accent character. Unfortunately, this interferes with hyphenation. Apart from this, the OT1 encoding has various other oddities, and is best avoided. OT1-encoded fonts often have a TeX name ending in `7t`.[4] Note that `ot1.enc` comes with afm2pl and is probably not available if you don't have afm2pl on your system.

T1 encoding

T1 is the successor to OT1. It uses all available slots, and has lots of accented characters, including those for Eastern European languages. Because the T1 encoding left no room for symbols such as '‰' or '©' or '‡' you will need to get those from a second encoding of the same font. This second encoding is called TS1 or 'text companion'.

[2]Omega and its offshoot Aleph are Unicode-based. Users may also be interested in XeTeX (`http://scripts.sil.org/xetex`), which is built on top of a regular TeX implementation and lets you use Unicode fonts installed on the system directly with TeX.

[3]If you don't specify an encoding, then you get the encoding from the `.afm` file, which is almost certainly not what you want.

[4]For afm2pl and afm2tfm, font names have no particular meaning. This is one more difference with fontinst. I add encoding suffixes such as `7t` and `8y` to font names just as reminders to myself.

For most traditional PostScript fonts, some of the accented characters in the T1 encoding aren't actually present and must be created with virtual font technology from a base character and an accent. Since it doesn't have to be done by TEX itself, this is no obstacle to hyphenation.

Although you can tell afm2pl to use T1 encoding, it can't create composite characters, and such composite characters will be missing unless they are already present in the original font.

T1-encoded fonts often have a TEX name ending in `8t`.

Texnansi encoding

The texnansi encoding, known as LY1 to LATEX, was introduced by Y&Y, the now-defunct company behind Y&YTEX, dviwindo and dvipsone. It combines a good selection of both accented letters and typographic symbols, and normally contains everything you need in a single encoding, at least for Western European languages. Texnansi-encoded fonts often have a name ending in `8y`.

The package texnansi selects the *texnansi* encoding and contains some additional code to smooth out incompatibilities with T1 and OT1.

A texnansi example

For this example, we choose Augie, a handwriting font from TEX Live. These are the commands for generating the `.tfm` and `.map` files:

```
afm2pl -p texnansi augie___.afm augie8y.pl
pltotf augie8y
```

This is `ly1augie.fd` (notice the `ly1` prefix):

```
\ProvidesFile{ly1augie.fd}
\DeclareFontFamily{LY1}{augie}{}
\DeclareFontShape{LY1}{augie}{m}{n}{
  <-> augie8y }{}
```

This is the LATEX code:

```
\documentclass{article}
\usepackage{texnansi}
\pagestyle{empty}
\pdfmapfile{=augie8y.map}

\newcommand{\fancyfont}%
  {\fontfamily{augie}\selectfont}

\begin{document}
\fancyfont
Hello, world!
Accents: \'el\`eve bl\"of \"i;
Symbols:
\textparagraph{} \textdaggerdbl{}
\texttrademark{} \textcopyright
\end{document}
```

And this is the result. Notice the extra symbols. These are absent from the T1 encoding and would have required a text companion font.

Hello, world!
Accents: élève blöf ï; Symbols: ¶ ‡ ™ ©

TrueType

Another scalable font format is TrueType, which is supported by pdftex but currently not by dvips. Font metrics are stored in the font file itself. Using TrueType is somewhat more work; the following commands are required to import a TrueType font such as Trebuchet:

```
ttf2afm trebuc.ttf >trebuc.afm
afm2pl -p texnansi trebuc trebuc8y
pltotf trebuc8y
<edit mapfile to replace .pfb with .ttf>
```

ttf2afm extracts the metric information from the `.ttf` file.[5]

afm2pl has no way of knowing that the `.afm` describes a TrueType font, and guesses that the actual fontfile is `trebuc.pfb`. Therefore you have to fix the mapfile afterwards.

We leave it as an exercise for the reader to write the `.fd` file and LATEX source for the following example:

Hello, world!
Accents: élève blöf ï; Kerning: WAV, LTa, WAV, LTa.
Symbols: ¶ ‡ ™ ©

Font-based uppercasing and letterspacing

afm2pl comes with an uppercased version *texnanuc* of texnansi. Uppercasing, *e.g.* in headings, works best in combination with letterspacing. For this, afm2pl has a parameter '`-m`'.

Warning: afm2pl implements letterspacing with kerns. Unfortunately, the `.tfm` format can contain only a limited number of kerns. If there are too many in the `.pl` file then all kerns and ligatures will be dropped from the generated `.tfm` file! So use this feature with care. fontinst implements letterspacing

[5]This will result in an empty encoding, unless you specify an encoding parameter. But we are going to ignore the encoding in the `.afm` anyhow.

by adding sidebearings via virtual fonts, and doesn't suffer from this limitation.

We can create a letterspaced, uppercased version of Trebuchet with the following commands:

```
ttf2afm trebuc.ttf >trebuc.afm
afm2pl -p texnanuc -m 100 trebuc trebucupp8y
pltotf trebucupp8y
<edit mapfile to replace .pfb with .ttf>
```

A corresponding fontfamily and fontshape declaration might look as follows:

```
\ProvidesFile{ly1trebuc.fd}
\DeclareFontFamily{LY1}{trebuc}{}
\DeclareFontShape{LY1}{trebuc}{m}{upp}{
  <-> trebucupp8y }{}
```

The fontshape upp for uppercasing is not an official LaTeX shape but that doesn't seem to matter. You can use the font as follows:

```
\documentclass{article}
\usepackage{texnansi}
\pagestyle{empty}
\pdfmapfile{=trebucupp8y.map}

\begin{document}
\fontfamily{trebuc}\fontshape{upp}
\selectfont
Letterspaced uppercasing
\end{document}
```

And this is the result:

L E T T E R S P A C E D   U P P E R C A S I N G

A font family

The next example uses a real font family, consisting of the usual four family members plus our letterspaced font. So we will need not only trebuc.ttf, as in the previous example, but also trebucbd.ttf, trebucit.ttf, and trebucbi.ttf. For each of these we'll have to run the ttf2afm — afm2pl — pltotf sequence, and we'll have to edit each of the generated map files, or create a combined mapfile.

Here is the `.fd` file:

```
\ProvidesFile{ly1trebuc.fd}
\DeclareFontFamily{LY1}{trebuc}{}
\DeclareFontShape{LY1}{trebuc}{bx}{n}{
  <-> trebucbd8y }{}
\DeclareFontShape{LY1}{trebuc}{m}{n}{
  <-> trebuc8y }{}
\DeclareFontShape{LY1}{trebuc}{bx}{it}{
  <-> trebucbi8y }{}
\DeclareFontShape{LY1}{trebuc}{m}{it}{
  <-> trebucit8y }{}
\DeclareFontShape{LY1}{trebuc}{m}{upp}{
  <-> trebucupp8y }{}
```

And this is the LaTeX code using it:

```
\documentclass{article}
\usepackage{texnansi}
\pagestyle{empty}
% better combine these mapfiles!
\pdfmapfile{=trebuc8y.map}
\pdfmapfile{=trebucbd8y.map}
\pdfmapfile{=trebucit8y.map}
\pdfmapfile{=trebucbi8y.map}
\pdfmapfile{=trebucupp8y.map}

\begin{document}
\fontfamily{trebuc}\selectfont
Hello, \textbf{world!}

Accents: \'el\`eve bl\"of \"i;
Kerning: WAV, LTa, \textit{WAV,
                          \textbf{LTa.}}

Symbols:
\textparagraph{} \textdaggerdbl{}
\texttrademark{} \textcopyright

\fontshape{upp}\selectfont
Letterspaced uppercasing
\end{document}
```

And this is the result:

Hello, **world!**
Accents: élève blöf ï; Kerning: WAV, LTa, *WAV, **LTa.***
Symbols: ¶ ‡ ™ ©
L E T T E R S P A C E D   U P P E R C A S I N G

Using dvips

If you go the dvips route, then you cannot use the `\pdfmapfile` macro. Instead, you have to enter additional mapfiles on the command line:

```
dvips -u +mapfile dvifile
```

The prefix + to the `mapfile` parameter is analogous to the = prefix for the `\pdfmapfile` macro: it tells dvips to use the named mapfile *in addition to* the default one.

Using afm2tfm

The original intention of afm2tfm was not to create fonts which are used directly by TeX. Instead, they were to serve as a basis for virtual fonts, *i.e.* recipes to compose fonts from other fonts. But it is not too difficult to subvert this intention. The less optimal way is to use the output of afm2tfm directly:

```
afm2tfm kuen408i -p texnansi.enc \
        quick.tfm >a2t.map
<edit a2t.map; see below>
```

Note that the `.afm` filename comes *before* the options. The `.enc` extension must be included.

The better way is to pretend to create a virtual font:

```
afm2tfm kuen408i -T texnansi.enc \
        -v slow.vpl null.tfm >a2t.map
vptovf slow.vpl
rm slow.vf
<edit a2t.map>
```

vptovf generates two files from `slow.vpl`, named `slow.vf` and `slow.tfm`. You should remove `slow.vf`, otherwise the dvi driver or pdftex would think that `slow` is a virtual font.

Mapfile information is written to standard output, which therefore has to be redirected, as shown above. It contains the following string:

```
quick Kuenstler480BT-Italic
  " TeXnANSIEncoding ReEncodeFont "
  <texnansi
```

(everything on one line). This has to be changed into:

```
slow Kuenstler480BT-Italic
  " TeXnANSIEncoding ReEncodeFont "
  <texnansi.enc <kuen408i.pfb
```

(also on one line).

The example below displays differences in spacing between the two: kerns and ligatures were only written to the `.vpl` file, not to `quick.tfm`.

Note. This is not an example for copying.

Hello, world!
Accents: élève blöf ï; Symbols: ¶ ‡ ™ ©;
Kerning: WAV, LTa
Kerning: WAV, LTa

Other options of afm2pl and afm2tfm

With both programs you can artificially slant, narrow and widen a font. afm2tfm can also generate artificial smallcaps. Such manipulated fonts rarely look good, though.

afm2pl also has some options for manipulating the ligkern table and for setting spacing parameters. For casual use, you don't need to bother with these.

OpenType

We are seeing more and more OpenType fonts, which are Unicode-based. These consist of either PostScript/Type 1 or TrueType outlines inside a TrueType wrapper. OpenType fonts may contain huge character sets, sometimes including smallcaps and oldstyle figures.

OpenType fonts with TrueType outlines have an extension `.ttf` and can be treated just like TrueType fonts.

OpenType fonts with Type 1 outlines have an `.otf` extension. You can get an `.afm` for an OpenType file by first converting it with FontForge to TrueType (tip from Taco Hoekwater):

```
fontforge -c 'Open($1); Generate($2);' \
          ofont.otf ofont.ttf
ttf2afm ofont.ttf >ofont.afm
afm2pl -p texnansi ofont ofont8y
pltotf ofont8y
<edit mapfile to replace '<ofont.pfb'
 with '<<ofont.otf'>
```

Note the <<, which means that the font is to be included in full. For commercial fonts, this is usually not allowed.

Or have a look at otftotfm, part of Eddie Kohler's LCDF Typetools and included in TEX Live.

Scripting

Various people have written scripts to automate font installation. ConTEXt users will be familiar with texfont, which, by the way, has an option to use afm2pl instead of afm2tfm.

Each example took several commands on a command line. So why not a script?

Actually, I did use scripts. But my scripts tend to be highly specific to the job at hand, and I keep them with those jobs. So it made more sense to me just to give the necessary commands, and let readers script their own solutions.

Font verification and comparison in examples

KAREL PÍŠKA
Institute of Physics, Academy of Sciences
182 21 Prague, Czech Republic
piska (at) fzu dot cz

Abstract

This contribution demonstrates several techniques for verifying and comparing fonts widely used with TEX: METAFONT fonts and outline fonts in the PostScript Type 1 and OpenType formats. The aim is to generate various proofsheet files in PDF or PostScript with node and control points, control vectors and hinting zones for the subsequent visual scanning of graphic glyph representation, calculations of differences between metric data (e.g. character widths), between contour curves for different versions or releases, etc., thus accomplishing the auditing process more quickly and efficiently. Numerous tools — METAFONT, METAPOST, (pdf)(LA)TEX, dvips, gv, FontForge, MetaType1, TEXtrace, mftrace, t1utils, awk, sed, sort and other programs — are used. Resulting differences "greater than negligible" often indicate problems with compatibility, sometimes they may signal a bug undetected even for a long time. The examples are mostly taken from the current TEX Live 2005. The results of verification of CM, EC, LM, CS and other fonts available from TEX Live or CTAN, comparison for compatibility and consistency and the information about differences and bugs will be reported.

Introduction

A short version of the article (low level font oriented and technical) is presented here. After a brief explanation of font elements important for typesetting with TEX (metrics and glyph images) we will show a limited number of illustrations. The motivation of the work was to prepare tools and intermediate results in a textual (lists, tables) and a visual form to find, detect and demonstrate differences, mistakes, cases of inconsistency and incompatibility and, in the next step, to improve past, current or future fonts.

Font types and font data

TFM character dimensions

The tfm (TEX font metric) files contain four dimensions for each character [1]:

- *charwd*, the width
- *charht*, the height above the baseline
- *chardp*, the depth below the baseline
- *charic*, the character's "italic correction"

These define the size of each character's "bounding box" which TEX needs to typeset. For formatting text TEX uses only metric information and does not need glyph shapes. The tfm files also contain information about *ligatures* and *kerning* pairs defining the space adjustment between two adjacent characters. The dvi output produced by TEX contains only references to glyphs. The real glyphs are absent in dvi and are included into the final output in PS/PDF by device drivers (e.g. dvips) or by pdfTEX. The tfm files can be converted by the tftopl program to human-oriented property list files. We can read, edit and process them in this text form more easily. afm is a metric format for Type 1 PostScript fonts. The tfm and afm formats represent and store metric data differently. The afm bounding box has a different meaning, and in afm the glyph width is defined by the WX parameter.

```
ec-lmr10.tfm/pl:
(CHARACTER C y
   (CHARWD R 0.5278)
   (CHARHT R 0.43055)
   (CHARDP R 0.194443)
   (CHARIC R 0.008)

lmr10.afm:
C 121 ; WX 527.77777 ; N y ; B 19 -205 508 431 ;
```

During our processing we check, compare and test for compatibility the metric data, especially the character widths, ligatures and kerning pairs crucial for typesetting, taking into account TEX's font limitations: A font contains at most 256 character codes, 255 different nonzero widths, at most 15 different nonzero heights, 15 different nonzero depths, and 63 different nonzero italic corrections.

Difference between TEX and outline fonts

The 'native' TEX font formats (`pk/tfm`) have no more than 256 characters, no character names, and any comparison using TEX is based on character sets defined by available encodings. On the other hand, the PostScript Adobe Type 1 fonts (`pfb/afm`) have glyph names, may contain many glyphs, but only 256 of which can be encoded, their encodings may be flexible, and all glyphs may be compared by names. Additionally, in OpenType (`otf`) many glyphs are encoded and available. Therefore, operations with an outline font could be independent of the TEX limitations and all glyphs present in the font can be processed.

Font tables and font specimens with TEX

To test completeness of a font's glyph set, we start with font tables and font specimens. For this task, `testfont.tex` (available from `macros/plain/base` on CTAN) from the basic TEX distribution, as well as `fonttabs` (`texmf/tex/csplain/fonttabs.tex`), and OFS [4] developed by Petr Olšák can be recommended.

METAFONT and bitmap fonts

METAFONT generates the metric `tfm` files and a bitmap representation of glyphs for a selected device ('mode'). The shapes of bitmap fonts are represented as a bitmap (a matrix of pixels) in *any* resolution. Unfortunately, probably no reference resolution exists. We run METAFONT and `dvips` with modes available from the widely distributed `modes.mf` (`CTAN: fonts/modes`):

```
mf '\mode='$MOD';' input font.mf
dvips -mode $MOD -D $RES
```

and with the corresponding resolutions, for example

```
MOD  300    600    1200    2400   2602   5333
RES   cx  ljfour  ljfzzz  supre  proof    crs
```

to test fonts for all designed sizes and also for various (low, middle and high) resolutions. There is no direct correlation between correctness or incorrectness of shapes in different design sizes or different resolutions (magnifications).

Outline fonts

Outline fonts, e.g. Adobe PostScript Type 1 and OpenType, are represented by their outline contour curves independent of resolution. Of course, rendering always depends on an output device for both outline and bitmap fonts. Good outline fonts that should be rendered properly elsewhere, for example, may be embedded in a PDF document and are more flexible than bitmap fonts because "rerendering" of a bitmap font for any device often causes a loss of quality. The aim of testing outlines is to verify *consistency* (font and glyph elements are unified for all fonts of a font family, preserved for all design sizes) and *compatibility* of versions, changes may be only improvements or corrections of mistakes (not producing new mistakes).

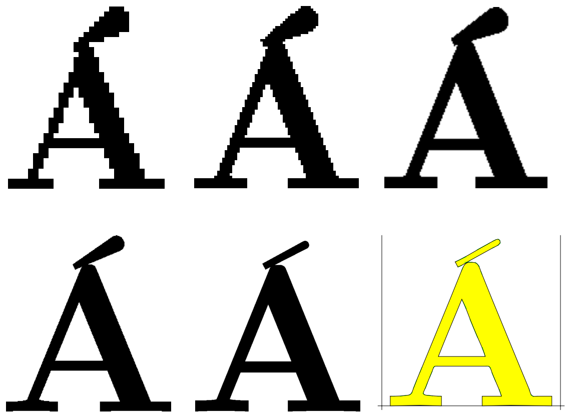


Figure 1: **csbx10 Á** at various resolutions; the shape of the accent changes.

Proofs of METAFONT fonts

csbx10: Á

Our approach is not to prove correctness of METAFONT programs but to check their products — glyphs in the `pk` format. Because it is impossible to choose one resolution to verify, we test the glyphs at various resolutions. Fig. 1 shows tests of **Á** from the **csbx10** font at following resolutions: 300, 600, 1200, 2602, and 5333 dpi. The last is the result of mftrace (autotracing bitmaps). We can detect a bug in CS fonts (from TEX Live 2005) — serif upper case accents depend on resolution (mode). We thus also observe a consequence of such bugs: the results of conversion to the outline font by autotracing high resolution bitmaps cannot be correct.

cmbx9/cmbx10: y

The next example illustrates behavior of **"y"** in the bold Computer Modern fonts (see Fig. 2 with 600,

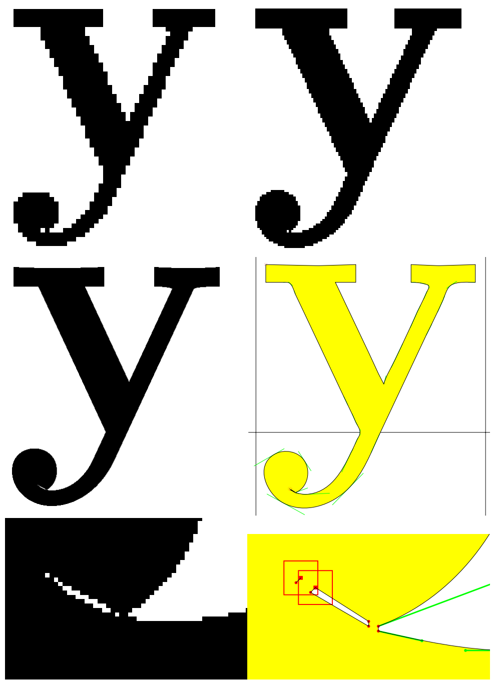


Figure 2: **cmbx10 y**: an artifact in the tail.

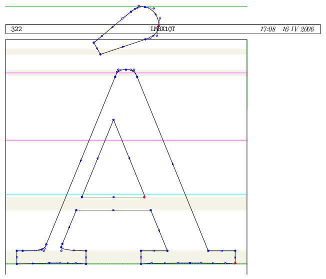


Figure 3: Standard proofsheet from MetaType1.

1200, 5333 dpi, and mftrace) where a strange scrap is generated. This bug in CM is very small, its occurrence is rare and well hidden (only in some sizes), unlike the previous easily evident bug in CS. A similar effect can be observed for cmbx9, cmbx7, cmbx8 (see also [2]). Probably, weaker correlations suppress this defect for cmbx5, cmbx6, cmbx12, and cmbx17.

The role of METAFONT today

METAFONT fonts may be simple or very complex, therefore debugging of a bitmapped glyph representation may be difficult. Although probably the users do not expect bitmap fonts in distributions today, METAFONT and METAPOST are still and will continue to be very important tools for font developers.

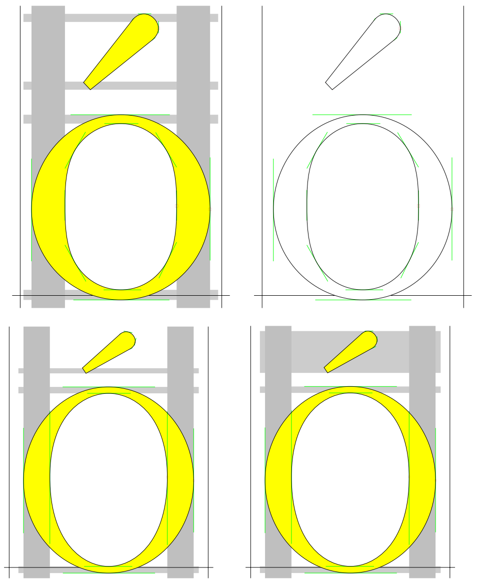


Figure 4: Outline font proofs.

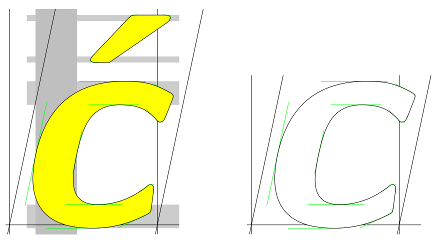


Figure 5: Extrema points and hints.

Proofs of outline fonts

Figure 3 shows the proofing page produced from the Latin Modern sources directly by programs from the MetaType1 package [3]. The following pictures in Fig. 4 demonstrate my own variants of proofsheets for screen and printer to recognize better the tinest details (using zoom) invisible in the previous figure because of mutual overlapping of such small details, e.g. the control points and vectors. Figures 4 and 5 also raise questions about an optimal approximation and a hinting strategy: To add the nodes at extrema or not, how to hint accents, etc.

FontForge [5], an open source font editor developed by George Williams, allows us to read and generate various font formats. We can also use it for checking and analyzing fonts, importing TEX font bitmaps, and exporting glyph outline curves in eps for subsequent processing.

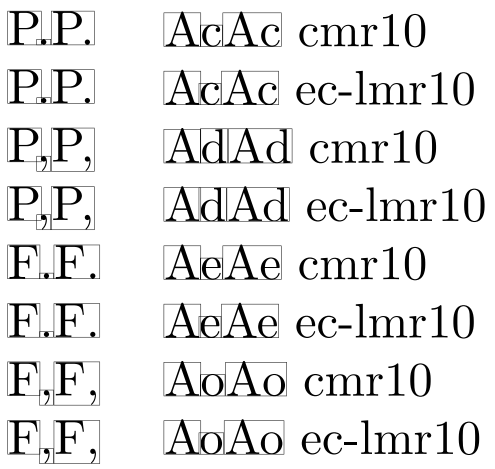


Figure 6: Kerning pairs in a visual form.

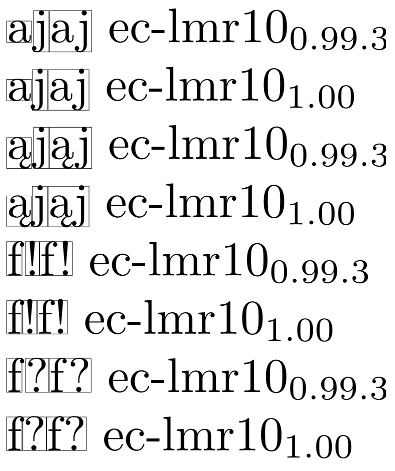


Figure 7: Kerning changes in LM.

Comparison of metric data

Analyzing the metrics we detect (automatically) cases of agreement or disagreement in dimension and kerning values. In Fig. 6 we present a small part of the comparison between the CM and LM `tfm` files in the T1 (ec-lm) encoding. `cmr10` and `ec-lmr10` are compatible in relation to the kerns `"P" : ",","."` and absence of kerns for `"F","T","V","W","Y" : ",","."`.

In `ec-lmr10` new kerning pairs have been introduced: `"A" : "c","d","e","o"`.

Fig. 7 demonstrates the changes between two versions of LM.

```
\newlength{\bbox}\newlength{\cbox}%
\def\fboxsep{0pt}\def\fboxrule{0.1pt}
\def\pair#1#2{%
  \settowidth{\bbox}{#1#2}%
  \settowidth{\cbox}{\mbox{#1}\mbox{#2}}%
  \addtolength{\bbox}{-\cbox}%
  \fbox{#1}\kern-0.2pt\kern\bbox\fbox{#2}%
  \fbox{#1#2}}
\pair{a}{j}
```

A short LaTeX macro `\pair` calculates the shift between two “kerned” boxes.

Comparison of glyph images

We will demonstrate two techniques:

- color mix with pdfTeX for visual scan
- outline comparison for outline fonts

Color mix with pdfTeX

Generating of comparative proofsheets using pdfTeX for mixing colors is possible for both bitmapped and outline fonts. Searching for differences needs subsequent human visual postprocessing. In our examples, the combination of red and cyan produces pink in the intersection. Here is the TeX code:

```
\usepackage{graphicx}
\def\Default{\pdfliteral{0 g 0 G}}
\pdfpageresources{/ExtGState
 << /Luminosity
    << /Type /ExtGState /BM /Luminosity >>
 >>}
\def\Acolor{1 0 0 rg}% red
\def\Bcolor{0 1 1 rg}% cyan

\renewcommand\C{\char#1}%
\Default  \fbox{\makebox[0pt][l]%
{\pdfliteral{/Luminosity gs \Acolor}\Afont\C}%
{\pdfliteral{\Bcolor}\Bfont\C}}%
```

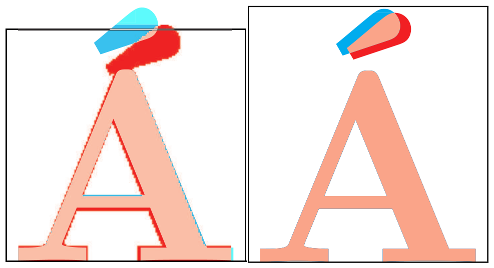


Figure 8: A color mix.

Fig. 8 demonstrates data from two samples:

left outline font `lmbx10` (ver. 0.99.3) [cyan] vs. bitmap font `csbx10` [red]

right lmbx10 (ver. 0.99.3) [cyan] vs. lmbx10 (ver. 1.00) [red] (both outline Type 1)

In a grayscale printing the red color looks dark, the cyan is lighter, and the intersection is the lightest.

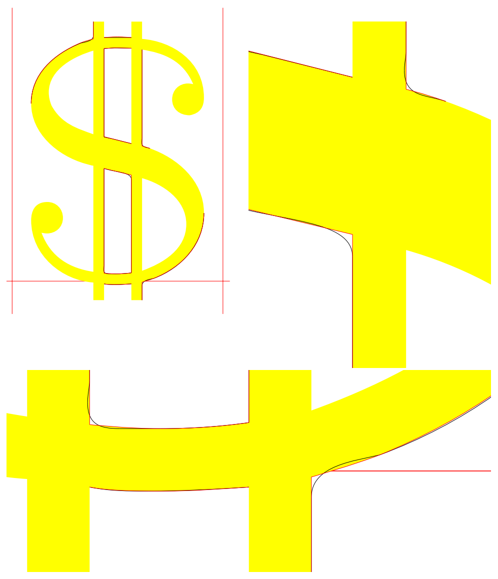


Figure 9: Improvement of outline approximation.

Comparison of outline fonts

Some elements of two outline fonts allow an automatic comparison. We can compare two fonts with a common glyph repertoire, e.g. to test two releases, alternatives, extensions or subsets of any font, or to test two similar fonts for differences. We compare all the glyphs available in both fonts by their names automatically and detect the following differences:

- presence and absence of a glyph with a given name (This may mean that a new glyph has been added or an old glyph has been removed, a glyph has been renamed, or sometimes, a glyph name may be invalid)
- different glyph shapes (at least one segment is different)
- different glyph widths (even after rounding to integer in the glyph coordinate space)

In the examples, the contour curves of the first (older) font are black. The contour curves of the second (newer) font are red and the filled glyph area is yellow. The glyph box frames in the older font are blue. The black and dark lines denote the older version in printing without colors.

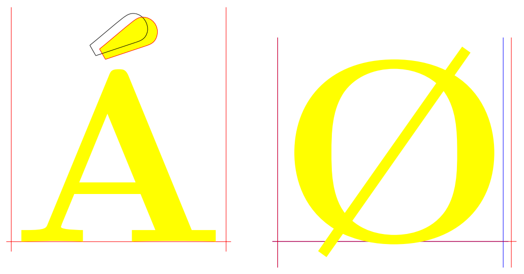


Figure 10: Modification and correction.

Figures 9 and 10 show differences between LM 0.99.3 and LM 1.00. In Fig. 9 the `dollaroldstyle` from `lmr10` has been improved (its conversion to outlines is better). Fig. 10 gives information about the modification of the acute accent and correction of glyph width in the `lmbx10` font.

Conclusion

Techniques extending and complementary to existing testing tools have been presented in various examples, to help font authors and maintainers in their time comsuming and expensive work. The results of tests were or may be applied as warnings, bug reports or suggestions. They were or are used for verification of the Type 1 version of public Indic fonts [6] available from CTAN and for tests of the LM fonts.

References

[1] Donald Knuth. *The METAFONTbook*.

[2] Bogusław Jackowski, Janusz M. Nowacki, Piotr Strzelczyk. “Programming PostScript Type 1 Fonts Using MetaType1: Auditing, Enhancing, Creating. *TUGboat* 24:3, pp. 575–581, *Proceedings of the XIV EuroTEX 2003 conference*, Brest, France, 24–27 June 2003.

[3] MetaType1 distribution. `ftp://bop.eps.gda.pl/pub/metatype1`.

[4] Petr Olšák. The OFS font management system. `ftp://matf.feld.cvut.cz/pub/olsak/ofs`

[5] George Williams. FontForge: an outline font editor. `http://fontforge.sourceforge.net`

[6] `CTAN:fonts/ps-type1/indic`

MlBibTEX meets ConTEXt

JEAN-MICHEL HUFFLEN
LIFC (FRE CNRS 2661)
University of Franche-Comté
16, route de Gray
25030 Besançon Cedex, France
hufflen (at) lifc dot univ dash fcomte dot fr
http://lifc.univ-fcomte.fr/~hufflen

Abstract

This article reports a first experiment of using MlBibTEX — our reimplementation of BibTEX — with ConTEXt, a TEX format more modern than LATEX. We show how to take as much advantage as possible of both ConTEXt and MlBibTEX features when they are used together. Also, many end-users are accustomed to using LATEX commands inside values of BibTEX fields, and such commands may be unrecognised by ConTEXt. We explain how patterns *and* preambles *allow us to solve such problems.*

Keywords

ConTEXt, `bib` module, bibliographies, bibliography styles, BibTEX, MlBibTEX.

Introduction

Listing the bibliographical references cited within a document can be done manually — if the LATEX word processor is used, that consists of typing successive `\bibitem` commands of a `thebibliography` environment [18, § 4.3.2] — but such an approach leads to texts difficult to maintain and reuse, because they are tightly bound to *bibliography styles*. A publisher or anthology editor might like authors' last names of a 'References' section to be typeset using small capitals, whereas another publisher would require the use of standard Roman letters for these last names. Likewise, first names may be abbreviated or put *in extenso*, w.r.t. the bibliography style used. In addition, doing a document's bibliography manually is error-prone: if this bibliography is *unsorted*, that is, if the order of items is the order of first citations of these items throughout the document, some change within the document's body can cause the bibliography to be reorganised. Likewise, keys based on the author-date system [19, § 12.3] may need to be recomputed if the bibliography is enriched.

A better method is to use a *bibliography processor*: such a program is given *citation keys*, searches bibliography database files for resources associated with these keys, and arranges them according to a bibliography style, the result being a source file for a 'References' section, suitable for a word processor. A well-known association between a word and bibliography processor is given by LATEX and BibTEX [21], working in tandem, although this example is not unique. As another example, Tib [1] has sometimes been used with *Plain TEX* [17]; more generally, other examples of bibliography processors are given in [25].

'Historically', BibTEX was initially designed to work with Scribe[1] [24]. In fact, only a few points related to TEX are hard-wired within BibTEX: using braces as delimiters, considering a group such as '`{\`*`command`*` ...}`' as an accent command applied to arguments in order to produce a single character [19, § 13.2.2], the use of the '~' character for unbreakable spaces when names are formatted [20, § 5.4], the `width$` function, provided by the style language, that returns the width of a string, expressed using TEX units [19, Table 13.8]. Thus bibliographic entries specified with BibTEX should be usable with any *format* built from TEX, provided that end-users do not put LATEX-specific commands inside field values. Let us recall that TEX basically provides a powerful framework

[1]That is why BibTEX uses the '`@`' character for specifying its commands and entry types: this character introduces a command name in Scribe, like '`\`' in TEX. This convention is also used within Texinfo [3], the GNU documentation format.

```
@BOOK{meaney2003,
        AUTHOR = {John Meaney},
        TITLE = {Context},
        PUBLISHER = {Bantam Books},
        YEAR = 2003,
        NUMBER = 2,
        SERIES = {The \emph{Nulapeiron}
                  Sequence},
        NOTE = {The Sequel to ‘‘Paradox’’.
                [Pas de version française
                 connue!] ! french},
        LANGUAGE = english}
```

Figure 1: Example of a bibliographical entry.

to format texts nicely, but to be fit for use, the definitions of this framework need to be organised in a format. The first formats were *Plain TEX* and LATEX; another format, more modern, is ConTEXt [5]. A bibliographic module, usable in conjunction with BibTEX, has been added to ConTEXt [6, 8]. This module defines ConTEXt commands to deal with the components (metadata) of bibliographical information.[2]

Over the last few years, we have designed and implemented a ‘new BibTEX’ — MlBibTEX, for ‘**M**ultiLingual BibTEX’. Of course, it has been designed to work with LATEX, but we plan to use it for other output formats, too [10]. This article is a revised and extended version associated with the presentation given at EuroTEX. It aims to report a first experiment of using MlBibTEX to build outputs suitable for ConTEXt.[3] First, we show how ConTEXt can be easily used to pretty-print bibliography database (.bib) files. Then we explain how an interface between ConTEXt and BibTEX can be improved when MlBibTEX is used. This article should be read without any difficulty by any user familiar with LATEX and BibTEX: it requires only basic knowledge of ConTEXt and its bib module. It also refers to some basic notions of XML[4] and Scheme, the implementation language for MlBibTEX.[5]

[2]If we compare this module to what is provided for LATEX, its approach is close to the jurabib package [19, § 12.5.1], in the sense that items of bibliographical information are given as arguments of new commands. If you would like to redefine the layout of a bibliography's items, just redefine these new commands.

[3]We have used the most recent version of ConTEXt at the time of writing, included in TEX Live 2005, available on DVD-ROM.

[4]e**X**tensible **M**arkup **L**anguage. Readers interested in an introductory book to this formalism can consult [23].

[5]The version used is described in [14].

```
<mlbiblio>
  ...
  <book id="meaney2003" language="english">
    ...
    <series>
      The <emph>Nulapeiron</emph> Sequence
    </series>
    <note>
      The Sequel to
      <emph emf="no" quotedf="yes">
        Paradox
      </emph>.
      <group language="french">
        Pas de version française connue!
      </group>
    </note>
  </book>
  ...
</mlbiblio>
```

Figure 2: XML tree for the bibliographical entry shown in Figure 1.

Pretty-print bibliographies

MlBibTEX's new syntactic features for bibliographical entries are detailed in [9]. Roughly speaking, any .bib file suitable for ‘old’ BibTEX can be processed by MlBibTEX, except that square brackets are ordinary characters for the former, syntactic delimiters for the latter. Figure 1 gives an example of a bibliographical entry for a book written in English (the value of the LANGUAGE field, handled by MlBibTEX). The value of the NOTE field includes a text to be put down only in French-speaking bibliographies, this text being enclosed by square brackets and labelled by the french language identifier.

As mentioned in [9], the result of parsing a .bib file can be viewed as an XML tree. For example, parsing a file containing the meaney2003 entry results in the XML tree sketched in Figure 2. Such a tree can be saved into a file and displayed *verbatim* or handled by tools belonging to XML's world. ConTEXt provides a way to handle XML texts [22], so it can deal with such files. Figure 3 sketches a pretty-printer for bibliographical entries by means of ConTEXt commands documented in [7, 22], other basic TEX commands — such as \expandafter or \uppercase — being documented in [17]. These bibliographical entries are displayed using MlBibTEX's syntax. In addition, MlBibTEX's new syntax for emphasising the parts of per-

```
\enableregime[il1]

\def\ProcessMlBibTeXFieldName[#1]{{\tt \expandafter\uppercase{#1} = \textbraceleft}}
\def\CloseMlBibTeXFieldValue{{\tt \textbraceright},\par}
\def\ProcessMlBibTeXLanguagePart[#1]{\PutLanguageCommand[#1]{\tt LANGUAGE =} #1,\par}
\def\ProcessMlBibTeXNamePart[#1]{{\tt #1 =>} }
\def\PutLanguageCommand[#1]{\doifelse{#1}{english}{\language[en]}{%
  \doifelse{#1}{french}{\language[fr]}{\doif{#1}{magyar}{\language[hu]}}}}

\defineXMLenvironment[mlbiblio] \startitemize \stopitemize
\defineXMLenvironment[book] {\item {\tt @BOOK\textbraceleft}\XMLpar{book}{id}{*unkeyed*},%
 \startnarrower[left] \ProcessMlBibTeXLanguagePart[\XMLpar{book}{language}{english}]} {%
 \stopnarrower {\tt \textbraceright}}
...
\defineXMLenvironment[author] {\ProcessMlBibTeXFieldName[author]} \CloseMlBibTeXFieldValue
...
\defineXMLenvironment[first] {\ProcessMlBibTeXNamePart[first]} {, }
\defineXMLenvironment[von] {\ProcessMlBibTeXNamePart[von]} {, }
\defineXMLenvironment[last] {\ProcessMlBibTeXNamePart[last]} \unskip
\defineXMLenvironment[junior] {, \ProcessMlBibTeXNamePart[junior]} \unskip

\defineXMLenvironment[asitis] {{\tt \textbraceleft}} {{\tt \textbraceright}}
\defineXMLenvironment[emph] {\doifelse{emph}{quotedf}{yes}{{\tt ‘‘}}{%
 \doifelse{emph}{emf}{yes}{{\tt \textbackslash emph\textbraceleft}\bgroup\em}{}}} {%
 \doifelse{emph}{emf}{yes}{{\tt \textbraceright}\egroup}{%
 \doifelse{emph}{quotedf}{yes}{{\tt ’’}}{}}}

\def\GroupMarker{! }
\defineXMLenvironment[foreigngroup] {{\tt [}%
 \bgroup\PutLanguageCommand[\XMLpar{foreigngroup}{language}{*error*}]} {%
 \egroup{\tt ] : \XMLpar{foreigngroup}{language}{*error*}} }
\defineXMLenvironment[group] {\startnarrower[left]{\tt [}%
 \bgroup\PutLanguageCommand[\XMLpar{group}{language}{*error*}]} {%
 \egroup{\tt ] \GroupMarker \XMLpar{group}{language}{*error*}} \stopnarrower}
\defineXMLenvironment[nonemptyinformation] {{\tt [}}\def\GroupMarker{* }} {}

\starttext
\processXMLfilegrouped{...}
\stoptext
```

Figure 3: Pretty-printing an XML tree resulting from parsing .bib files.

son names [21, § 4], based on keywords, is used. For example:

AUTHOR = {first => John, last => Meaney},

We can notice that keywords, syntactic delimiters, and field names are typeset using a typewriter font, whereas the Roman typeface is used for metadata. As another pretty-printing feature, typographical effects are put into action. For example, the value of the SERIES field will be rendered as follows:

SERIES = {The \emph{*Nulapeiron*} Sequence},

Likewise, any information is typeset using the typographical conventions of its own language:

NOTE =
{ ... [Pas de version française connue !] ... }

where the exclamation mark is preceded by a thin space character, as in French.

If we look at the text given in Figure 3, we notice that the only heavy part concerns *language identifiers*. ConTEXt uses ISO codes for languages [5, Ch. 7] and can switch to any language via the \language[...] command, without any preliminary declaration such as LATEX needs when the babel package is loaded [19, § 9.2]. MlBibTEX's language identifiers are unambiguous prefixes of packages or options of the babel package, as explained in [12]. For example:

- polish is for the option of the babel package,
- polski is for the polski package [4, App. F],
- pol is for either of these last two, the final choice depends on what a user puts in the document's preamble,
- po is ambiguous because it may be the prefix of '*Po*lish' or '*Po*rtuguese'.

In fact, we need a complete correspondence table, with our \PutLanguageCommand command given in Figure 3 implementing it only partially.[6] Let us remark that such a correspondence table could be useful for other purposes, e.g., generating bibliographies for documents written using DocBook[7] [27]. This table between the structure managing MlBibTEX's language identifiers [12] and ISO codes for languages [2] has been implemented within the Scheme functions of MlBibTEX.[8]

Figure 3 gives a rough version of such a pretty-printer, which may be improved.[9] For example, the error cases are just labelled within the source text by identifiers surrounded by '*', which could be refined into a more efficient marking of errors. We can also align '=' signs vertically between field names and values. That can be done in a tabulate environment, but leads to slighly complicated ConTEXt commands, because we need to collect the content of a table before formatting it.

ConTEXt and MlBibTEX together

If you would like BibTEX to generate specifications of publications suitable for ConTEXt from bibliographical entries, you may use the \setupbibtex command, as explained in [8, § 2.4]. This command gets access to bibliography styles suitable for ConTEXt, that is, handled by the bib module. Since MlBibTEX can process bibliography styles using the bst language [20] in compatibility mode [13], it can deal with these styles. However, we do not recommend this solution, which should be temporary, from our point of view. In addition, the compatibility mode is not very efficient for sake of implementation issues.[10] A first improvement could be the development of new bibliography styles, using the nbst[11] language, close to XSLT[12] [26] and described in [9].

As mentioned in [19, § 13.5.2], the choices among different styles for displaying person names, work titles, ... causes a combinatorial explosion. Besides, all the functions of a bibliography style of BibTEX must be grouped into a unique file, so a rich library of bibliography styles for BibTEX should include a huge number of styles, each being monolithic. As explained in [11], several fragments of a bibliography style written using the nbst language can be assembled dynamically, provided that there is no conflict among the templates programmed using nbst. Consequently, designing styles according to a modular approach is easier in MlBibTEX than in BibTEX. Moreover, an extended version of the \setupbibtex command could allow the use of *several complementary files* for a bibliography style.

ConTEXt vs LATEX

End users sometimes put LATEX commands within the values of BibTEX fields. Some commands aim to increase the expressive power of the information put into .bib files, an example being given by the value of the SERIES field in Figure 1. Other examples are related to some features of BibTEX:

```
{Maria {\MakeUppercase{d}e La} Cruz}
```

— given in [19, p. 767] about person names — allows BibTEX to interpret '{de La}' as a particle,[13] because this group, surrounded by braces, begins with a lowercase letter, even if this particle should be typeset as 'De La'. Some commands are recognised by ConTEXt, some not. There are two solutions to this problem:

- when outputs for ConTEXt are produced, the contents of @preamble rubrics included in .bib files

[6]In fact, this \PutLanguageCommand command could be written in an easier way, since complete names such as english, french, ... also work as arguments of the \language command of ConTEXt. However, this feature is not described in [5].

[7]DocBook is an XML-based system for writing structured documents.

[8]How to put this implementation into action is shown in Figure 4.

[9]A more elaborated version can be downloaded from MlBibTEX's home page: http://lifc.univ-fcomte.fr/~hufflen/texts/mlbibtex/contextstuff/.

[10]When MlBibTEX parses a .bib file, it tries to organise information into a deep tree, as far as possible. For example, the components of a person name are split into subtrees. When the compatibility mode is used, these components are serialised into a string, and destructured again by the format.name$ function of bst [20].

[11]**N**ew **B**ibliography **ST**yles.

[12]e**X**tensible **S**tylesheet **L**anguage **T**ransformations, the language of transformations used for XML texts.

[13]'Maria' being the first name, 'Cruz' the last name.

```
(and-let* ((((log-output-p-pv 'open) jobname))                   ; Opening the log (.mblg) file.
           (((bibtexkey-alist-pv 'add-key) "hoekwater2001"))    ; Citation key to be processed.
           ...
           (((bibtexkey-alist-pv 'extend)))    ; If we want to process all the entries (\nocite{*}).
           ((let ((bib-suffix ".bib"))
              (every (lambda (filename)    ; Parsing .bib files. If the suffix is not given, the filename-plus
                                           ; function adds it.
                       (s-parse-bib-file (filename-plus filename bib-suffix #f)))
                     bibliographyfilename-list)))
           (sxml-mlbiblio-tree (s-get-sxml-mlbiblio-tree)) ; Build the SXML tree.
           (((language-trie-pv 'use-iso-code-table))) ; Using ISO codes for all the languages.
           (((preamble-pv 'set) "contextpreamble"))   ; Using @contextpreamble{...} as preambles.
           (k1 (n-assemble-nstyles stylefilename-list)) ; Styles are assembled and compiled into a
                                                        ; so-called k1 function.
           (((output-encoding-pv 'set) 'latin1)) ; Accented letters of Latin-1 allowed in the output file.
           (((bbl-output-p-pv 'open) jobname)))  ; Opening the output file.
  (k1 sxml-mlbiblio-tree)                        ; Applying the whole style to the SXML tree.
  ((bbl-output-p-pv 'close))                     ; Closing files.
  ((log-output-p-pv 'close))
  #t)                                            ; Final result.
```

Figure 4: MlBibTEX's kernel for use with ConTEXt.

are not written as BibTEX would do; instead, MlBibTEX uses `@contextpreamble` rubrics,[14] which can be used to implement some LATEX commands in ConTEXt; switching to another preamble is controlled by an option of the MlBibTEX program;[15]

- a better solution is given by *patterns*, expressed in Scheme, replacing substrings by XML-conformant strings; for example:[16]

  ```
  (define-pattern "\\emph{#1}"
    "<emph>#1</emph>")
  ```

  Patterns aim to process any LATEX command included in a `.bib` file, including user-defined commands, as explained in [10]. 'General' patterns are planned for the next version, only some predefined patterns are implemented now, mostly for letters accented by means of TEX commands.[17]

[14]This new command does not interfere with parsing `.bib` files by 'old' BibTEX, because it looks like:
`@...{⟨string⟩ (# ⟨string⟩)*}`
where '⟨*string*⟩' is surrounded by braces or double quotes. Such a command is ignored by 'old' BibTEX.

[15]... or see how to process in Scheme in Figure 4.

[16]Let us recall that in Scheme, the '\' character is used to escape special characters in constant strings. To include it within a string, it must be itself escaped.

[17]The internal representation uses Latin-1, accented letters of this encoding being viewed as single characters.

This solution is more general, not limited to bibliographies usable by ConTEXt. Let us assume that you have to convert a `.bib` file into HTML,[18] and consider the following title:

```
\ConTeXt, the Manual
```

Even if displaying '`\ConTeXt`' on a Web page does not cause any error, it is better to introduce this pattern:

```
(define-pattern "\\ConTeXt"
  "<symbol name='ConTeXt'/>")
```

Now you can define a way to display this symbol within a bibliography style.

Direct interface

ConTEXt does not deal with the same auxiliary files as LATEX. Moreover, it builds an `.aux` file only if the `\setupbibtex` command is activated. Let us recall that BibTEX reads only `.aux` files, never `.tex` files. However, MlBibTEX may need to parse the preamble of a source file, as explained in [12]. Concerning outputs suitable for ConTEXt, the information of interest is the encoding: can MlBibTEX put accented letters of Latin-1 directly into the resulting file? Or does it have to use TEX accent commands?

[18]**H**yper**T**ext **M**arkup **L**anguage.

A better solution than making useless .aux files and the parsing of preambles of ConTEXt documents (parts between the beginning of a document and the \starttext command) is to build a *driver* directly written in Scheme. Of course, this task requires some knowledge of both the Scheme programming language and the broad outlines of MlBibTEX's implementation, but the result is a small-sized program, as shown in Figure 4. You can see how to add a citation key as if it were caught from an .aux file, and how to get all the entries of .bib files as the \nocite{*} command of LATEX would cause to. We also show how to use a preamble command — @...{...} — specific for ConTEXt. Other information to be supplied is:

jobname (string) the base name of the main input file processed by ConTEXt;

bibliographyfilename-list (string list) all of the .bib file names to be searched;

stylefilename-list (string list) all the fragments of a bibliography style.

The and-let* macro [15] causes the sequential evaluation of the clauses of its first argument to be stopped as soon as a false value (for a failure) is returned. For example, the evaluation of the whole expression given in Figure 4 stops and returns the false value if a log file (.mblg file) cannot be opened. Otherwise, the non-false result of a clause may become the value of a local variable. For example, the sxml-biblio-tree variable is given the bibliographical entries in the SXML[19] format. Then the other arguments of the and-let* macro are evaluated sequentially if all the clauses succeed, that is, if there is no error in parsing .bib files and building the bibliography style. In our case, the style — which results in a Scheme function — is applied to the bibliographical entries, and output files are closed. Finally, the true value is returned.

Going further, the texexec script, which launches successive run phases of ConTEXt, could be extended to launch the MlBibTEX program.

Conclusion

When we began this task, we had written only some small-sized examples using ConTEXt and emphasising its differences with LATEX. And we were afraid we would have to reprogram some important parts of MlBibTEX. To be honest, changes were needed, but not as many as we believed. Concerning the bib module, we learned it more quickly than we planned. The first meeting between MlBibTEX and ConTEXt has succeeded.

[19]Scheme implementation of XML. See [16] for more information.

Acknowledgements

Many thanks to Hans Hagen and Taco Hoekwater, who kindly answered my very ConTEXt-nical questions. I am also grateful to Karl Berry, who proofread this article.

References

[1] James C. Alexander: *Tib: a TEX Bibliographic Preprocessor*. Version 2.2, see CTAN: biblios/tib/tibdoc.tex. 1989.

[2] Harald Tveit Alvestrand: *Request for Comments: 1766. Tags for the Identification of Languages*. UNINETT, Network Working Group. March 1995. http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc1766.html.

[3] Robert J. Chassell and Richard M. Stallman: *Texinfo. The GNU Documentation System. Version 4.8*. http://www.gnu.org/software/texinfo. December 2004.

[4] Antoni Diller: *LATEX wiersz po wierszu*. Wydawnictwo Helio, Gliwice. Polish translation of *LATEX Line by Line* with an additional annex by Jan Jelowicki. 2001.

[5] Hans Hagen: *ConTEXt, the Manual*. November 2001. http://www.pragma-ade.com/general/manuals/cont-enp.pdf.

[6] Taco Hoekwater: "The Bibliographic Module for ConTEXt". In: *EuroTEX 2001*, pp. 61–73. Kerkrade (the Netherlands). September 2001.

[7] Taco Hoekwater: *ConTEXt System Macros. Part 1: General Macros*. 2002. http://tex.aanhet.net/context/syst-gen-doc.pdf.

[8] Taco Hoekwater: *ConTEXt. Module Documentation*. March 2006. http://dl.contextgarden.net/modules/t-bib/doc/context/bib/bibmod-doc.pdf.

[9] Jean-Michel Hufflen: "MlBibTEX's Version 1.3". *TUGboat*, Vol. 24, no. 2, pp. 249–262. July 2003.

[10] Jean-Michel Hufflen: "MlBibTEX: beyond LATEX". In: Karl Berry, Baden Hughes and Steven Peter, eds., *Preprints for the 2004 Annual Meetings*, pp. 77–84. TUG, Xanthi, Greece. August 2004.

[11] Jean-Michel Hufflen: "Making MlBibTEX Fit for a Particular Language. Example of the Polish Language". *Biuletyn GUST*, Vol. 21, pp. 14–26. 2004.

[12] Jean-Michel Hufflen: *Managing Languages within MlBibTEX*. To appear. June 2005.

[13] Jean-Michel Hufflen: "BibTEX, MlBibTEX and Bibliography Styles". *Biuletyn GUST*, Vol. 23, pp. 76–80. In *BachoTEX 2006 conference*. April 2006.

[14] Richard Kelsey, William D. Clinger, Jonathan A. Rees, Harold Abelson, Norman I. Adams IV, David H. Bartley, Gary Brooks, R. Kent Dybvig, Daniel P. Friedman, Robert Halstead, Chris Hanson, Christopher T. Haynes, Eugene Edmund Kohlbecker, Jr, Donald Oxley, Kent M. Pitman, Guillermo J. Rozas, Guy Lewis Steele, Jr, Gerald Jay Sussman and Mitchell Wand: "Revised^5 Report on the Algorithmic Language Scheme". *HOSC*, Vol. 11, no. 1, pp. 7–105. August 1998.

[15] Oleg B. Kiselyov: `and-let*`*: an* `and` *with local bindings, a guarded* `let*` *special form*. March 1999. `http://srfi.schemers.org/srfi-2/`.

[16] Oleg E. Kiselyov: *XML and Scheme*. September 2005. `http://okmij.org/ftp/Scheme/xml.html`.

[17] Donald Ervin Knuth: *Computers & Typesetting. Vol. A: The TEXbook*. Addison-Wesley Publishing Company, Reading, Massachusetts. 1984.

[18] Leslie Lamport: *LATEX. A Document Preparation System. User's Guide and Reference Manual*. Addison-Wesley Publishing Company, Reading, Massachusetts. 1994.

[19] Frank Mittelbach and Michel Goossens, with Joannes Braams, David Carlisle, Chris A. Rowley, Christine Detig and Joachim Schrod: *The LATEX Companion*. 2nd edition. Addison-Wesley Publishing Company, Reading, Massachusetts. August 2004.

[20] Oren Patashnik: *Designing BibTEX Styles*. February 1988. Part of the BibTEX distribution.

[21] Oren Patashnik: *BibTEXing*. February 1988. Part of the BibTEX distribution.

[22] PRAGMA ADE, `http://www.pragma-ade.com/general/manuals/example.pdf`: *XML in ConTEXt*. November 2001.

[23] Erik T. Ray: *Learning XML*. O'Reilly & Associates, Inc. January 2001.

[24] Brian Keith Reid: *SCRIBE Document Production System User Manual*. Technical Report, Unilogic, Ltd. 1984.

[25] David Rhead: *The "Operational Requirement" (?) for Support of Bibliographic References by LATEX 3*. Technical Report L3–005, LATEX 3. August 1993.

[26] W3C: *XSL Transformations (XSLT). Version 1.0*. W3C Recommendation. Edited by James Clark. November 1999. `http://www.w3.org/TR/1999/REC-xslt-19991116`.

[27] Norman Walsh and Leonard Muellner: *DocBook: The Definitive Guide*. O'Reilly & Associates, Inc. October 1999.

Appendix G illuminated

BOGUSŁAW JACKOWSKI
BOP s.c., Gdańsk, Poland
`B_Jackowski (at) gust dot org dot pl`

Introduction

This paper aims to provide a collection of illustrations to *Appendix G* of *The TEXbook* [1].

To begin with, I will summarize briefly the main issues of *The TEXbook* which will be dealt with here; next, I confine myself to the explanation of the figures. Naturally, I will use the same notation as is used in *Appendix G*.

I recommend reading this paper simultaneously with *Appendix G*, although they partly overlap.

Motivation

TEX's algorithm for typesetting mathematical formulas is precisely described by Donald E. Knuth in *Appendix G* of *The TEXbook*. The description suffices to implement the algorithm in other languages. For example, it was implemented in JavaScript by Davide P. Cervone [2].

The only drawback of *Appendix G* is that no illustrations are provided. Of course, it is only a relative drawback. Professor Knuth apparently can live without illustrations. My comprehension critically depends on pictures. When they are missing in the original text, I end up making sketches while reading.

A few years ago, during my umpteenth reading of *Appendix G*, I prepared a bunch of sketches for myself. I didn't think about publishing them, as I was convinced that it is just my predilection or idiosyncrasy; but, judging from the paucity of available math fonts, I concluded that perhaps others might have similar problems.

Therefore, I decided to publish my illustrations to *Appendix G* in hope that they may prove useful, for example, for those working on math extensions for the available non-math fonts. Moreover, they may turn out to be helpful in future works on the improvement of TEX; after all, the algorithm is older than a quarter of century, and the world is not sleeping. For example, Murray Sargent III from Microsoft published recently (April, 2006) an interesting note on using Unicode for coding math [3]. He apparently was inspired by TEX: the notation is certainly TEX-based and well-known names appear in the acknowledgements and bibliography (Barbara Beeton, Donald E. Knuth, Leslie Lamport).

Math styles

In math formulas, the following eight styles are used:

- D, D' — in display formulas, generated out of text placed between double dollars `$$...$$` (*display style*);
- T, T' — in formulas occurring in a paragraph, i.e., placed between single dollars `$...$` (*text style*);
- S, S' — in formulas occurring in lower or upper indices, i.e., after the symbols `^` and `_` (*script style*);
- SS, SS' — in formulas occurring in indices of indices or deeper (*scriptscript style*).

Typically, in the plain format, 10-point fonts are used for the styles D and T, 7-point fonts for the style S, and 5-point fonts for the style SS. The "primed" styles, called cramped, use the same point sizes; they differ from the uncramped ones in the placement of subformulas. Table 1 defines the relations between styles of formulas and their subformulas.

The symbol C will denote the current style, the symbol $C\uparrow$ the corresponding style of a superscript, and the symbol $C\downarrow$ the corresponding style of a subscript. From the rules given in Table 1 it follows that $C\downarrow = (C\uparrow)'$.

Math lists

When TEX reads math material, it makes a *math list* out of it. The math list contains the following math-specific objects:

Basic style	Superscript style	Subscript style
D, T	S	S'
D', T'	S'	S'
S, SS	SS	SS'
S', SS'	SS'	SS'

Basic style of the formula α `\over` β	Numerator (α) style	Denominator (β) style
D	T	T'
D'	T'	T'
T	S	S'
T'	S'	S'
S, SS	SS	SS'
S', SS'	SS'	SS'

Table 1: Rules of the style change — indices (top) and fractions (bottom)

- *atom*, the basic element;
- *generalized fraction*, the result of `\above`, `\over`, etc.;
- *style change*, the result of `\displaystyle`, `\textstyle`, etc.;
- *boundary element*, result of `\left` or `\right`;
- *4-way choice*, the result of `\mathchoice`.

There are several types of atoms, depending on their contents. Table 2 (a duplicate of the table given on page 158 of *The TEXbook*) lists all possible cases.

Moreover, the math list may contain elements specific to a vertical list:

- *horizontal material* (*rules*, *penalties*, *discretionaries* or *whatsits*);
- *vertical material*, inserted by `\mark`, `\insert` or `\vadjust`;
- *horizontal space*, inserted by `\hskip`, `\kern`, or (acceptable only in math mode) `\mskip`, `\nonscript` or `\mkern`.

The math list is processed twice; after the second pass the "normal" vertical list is created. The scheme of the algorithm (consisting of 22 steps) is shown in Figure 1.

As we will see, font dimension parameters play an essential role. Following *The TEXbook*, I will denote the $i^{\rm th}$ parameter of the second family (that is, `\textfont2`, `\scriptfont2`, `\scriptscriptfont2`) by σ_i, and the $i^{\rm th}$ parameter of the third family (`\textfont3`, `\scriptfont3`, `\scriptscriptfont3`)

Name	Description of the atom
Ord	ordinary, e.g., x
Op	holding a big operator, e.g., $\sum$
Bin	holding a binary operator, e.g., $+$
Rel	holding a relational symbol, e.g., $=$
Open	holding an opening symbol, e.g., (
Close	holding an closing symbol, e.g.,)
Punct	holding a punctuation symbol, e.g., ,
Inner	"inner", e.g., $\frac{1}{2}$
Over	holding an overlined symbol, e.g., $\overline{x}$
Under	holding an underlined symbol, e.g., $\underline{x}$
Acc	holding an accented symbol, e.g., $\hat{x}$
Rad	holding a radical symbol, e.g., $\sqrt{2}$
Vcent	holding a *vbox* produced by `\vcenter`

Table 2: The types of atoms which may appear in a math list

by ξ_i. Note that the σ_i parameters have different values for different styles, while the values of the ξ_i parameters do not depend on the current style — when Computer Modern fonts and the plain format are used.

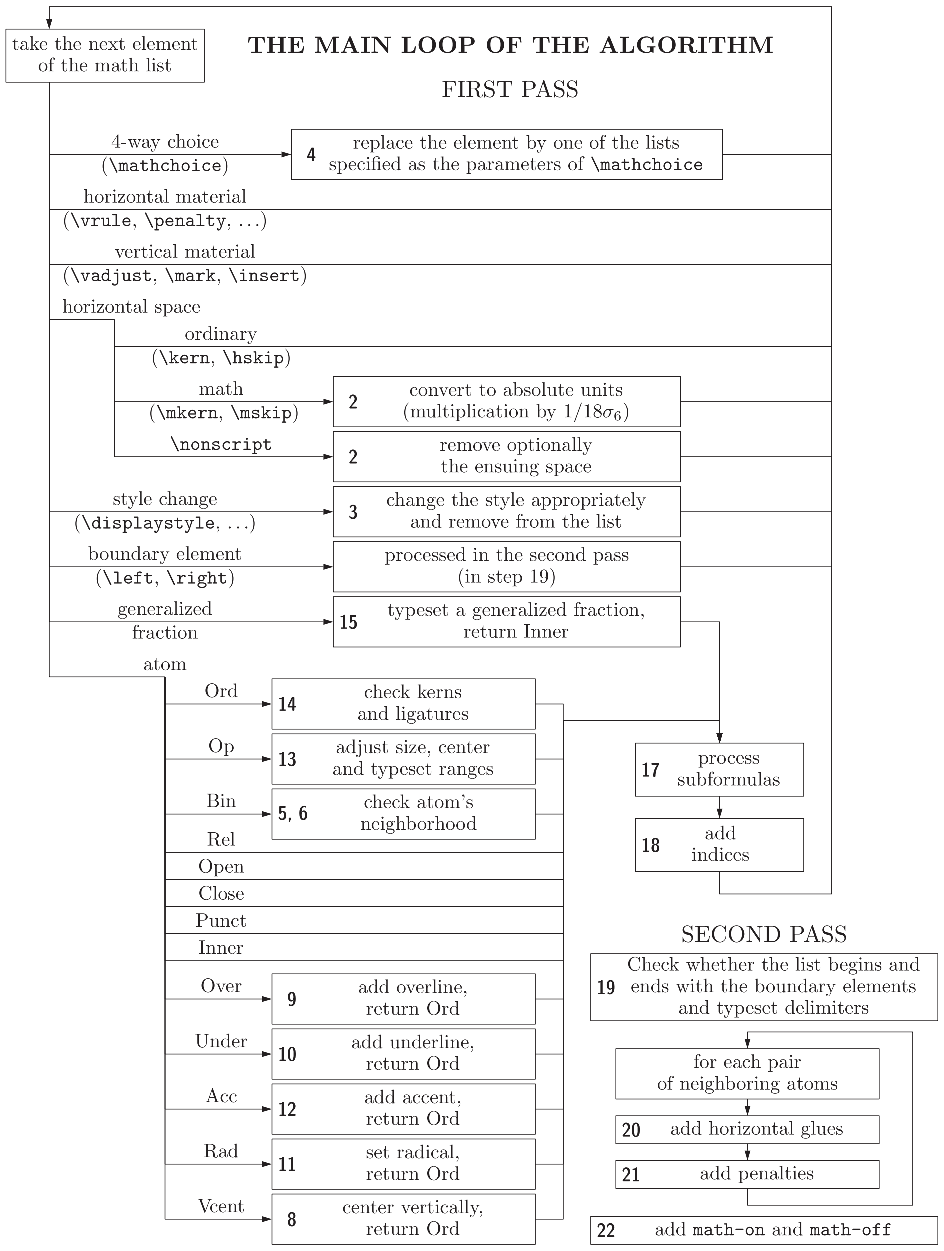
Selected steps of the algorithm

In most cases, the steps of the algorithm are straightforward and a short verbal explanation suffices. More detailed elaboration is needed, in my estimation, only for the typesetting of: radicals (step 11), mathematical accents (step 12), operators with limits (step 13), generalized fractions (step 15), and formulas with indices (step 18). The ensuing subsections deal with these steps.

Typesetting radicals

The process of assembling a formula containing a radical is presented in Figure 2. The top part of the picture shows the components (the radicand is typeset using the C' style), the bottom part the result of the assembling. Let w_x, h_x, d_x denote respectively the width, the height and the depth of the radicand, and w_y, h_y, d_y those of the radical symbol. The height of the radical symbol is expected to be equal to the thickness of the math rule, i.e., $h_y = \theta = \xi_8$. (A font designer may decide that $h_y \neq \xi_8$ but I cannot see the rationale for such a decision.) The quantity ψ is defined as follows:

$$\psi = \begin{cases} \xi_8 + \frac{1}{4}\sigma_5, & \text{styles } D, D', \\ \frac{5}{4}\xi_8, & \text{other styles.} \end{cases}$$



take the next element of the math list
THE MAIN LOOP OF THE ALGORITHM
FIRST PASS
4-way choice (\mathchoice)
4 replace the element by one of the lists specified as the parameters of \mathchoice
horizontal material (\vrule, \penalty, ...)
vertical material (\vadjust, \mark, \insert)
horizontal space
ordinary (\kern, \hskip)
math (\mkern, \mskip)
2 convert to absolute units (multiplication by 1/18σ6)
\nonscript
2 remove optionally the ensuing space
style change (\displaystyle, ...)
3 change the style appropriately and remove from the list
boundary element (\left, \right)
processed in the second pass (in step 19)
generalized fraction
15 typeset a generalized fraction, return Inner
atom
Ord
14 check kerns and ligatures
Op
13 adjust size, center and typeset ranges
Bin
5, 6 check atom's neighborhood
Rel
Open
Close
Punct
Inner
Over
9 add overline, return Ord
Under
10 add underline, return Ord
Acc
12 add accent, return Ord
Rad
11 set radical, return Ord
Vcent
8 center vertically, return Ord
17 process subformulas
18 add indices
SECOND PASS
19 Check whether the list begins and ends with the boundary elements and typeset delimiters
for each pair of neighboring atoms
20 add horizontal glues
21 add penalties
22 add math-on and math-off

Figure 1: The scheme of the algorithm of the math list processing; the numbers at the left of the boxes refer to the steps of the algorithm, as described in *Appendix G*

The quantity Δ is computed in such a way that the radical symbol is vertically centered with respect to the radicand: $\Delta = \frac{1}{2}\big(d_y - (h_x + d_x + \psi)\big)$.

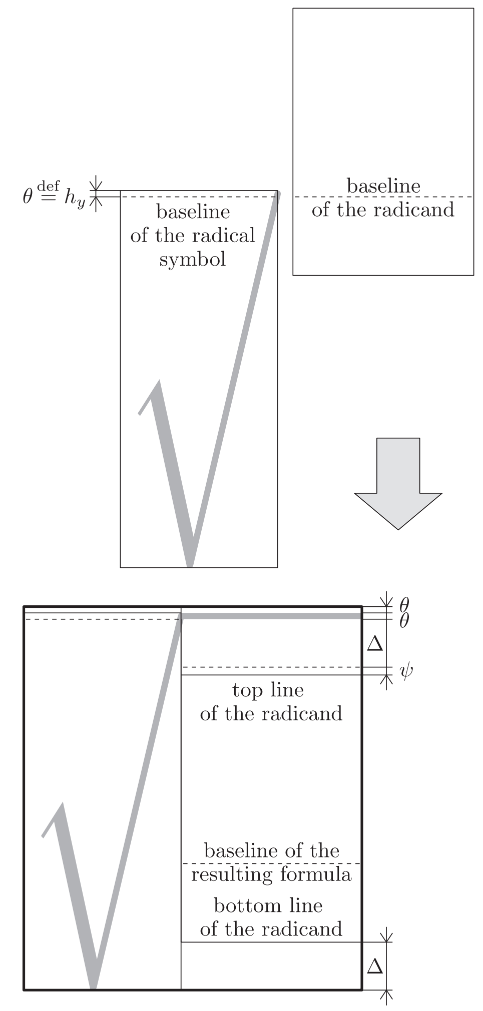
The baseline of the resulting formula coincides with the baseline of the radicand.

Typesetting mathematical accents

The typesetting of an accented formula is simpler than the typesetting of a radical. Nevertheless, Figures 3 and 4 reveal non-trivial subtleties of the routine.

As before, let w_x, h_x, d_x denote respectively the width, the height and the depth of the accentee (typeset in the style C'), and w_y, h_y, d_y those of the accenter. Actually, these are the widths of the respective boxes; if the accentee is a symbol, its width, w_x, is computed as the sum $wd + ic$, where wd is the nominal (metric) width of the accentee, and ic the italic correction.

Both the accenter and accentee boxes are put into a *vbox* one above the other, and a negative vertical



$\theta \overset{\text{def}}{=} h_y$
baseline of the radical symbol
baseline of the radicand
θ
θ
Δ
ψ
top line of the radicand
baseline of the resulting formula
bottom line of the radicand
Δ

Figure 2: Assembling a radical; symbols explained in the text

kern, $-\delta$, is inserted between the boxes, where $\delta = \min(\textit{x-height}, h_x)$. The *x-height* is defined by the fifth dimen parameter (`\fontdimen5`) of the accenter font.

The horizontal shift of the accenter, s, is equal to the implicit kern between the accentee and the special character, *skewchar* (defined by the command `\skewchar`); in the plain format, it is the character of code 127 (*tie after*) for family 1, and the character of code 48 (*prime*) for family 2. The kern has nothing to do with the shape of the `\skewchar`, but is intended to provide an appropriate correction due to the skewness of the accentee. If the accentee is already a boxed formula, TEX assumes that $s = 0$.

The width of the resulting formula is always equal to the width of the accentee, w_x; the baseline of the resulting formula coincides with the baseline of the accentee.

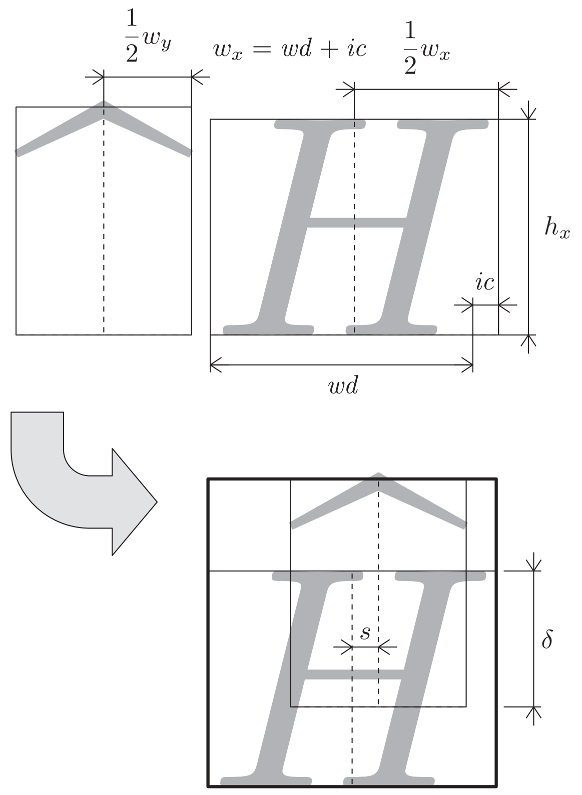


Figure 3: Assembling an accented formula, $w_y \leq w_x$; symbols are explained in the text

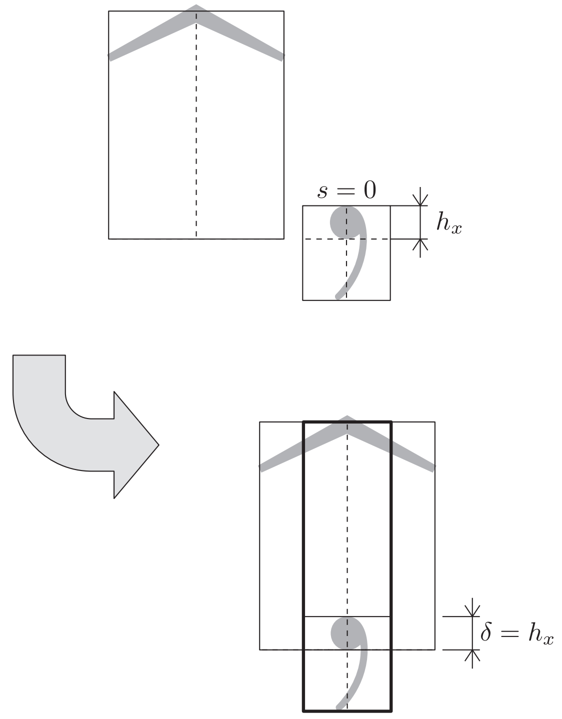


Figure 4: Assembling an accented formula, $w_y > w_x$; symbols have the same meaning as in Figure 3

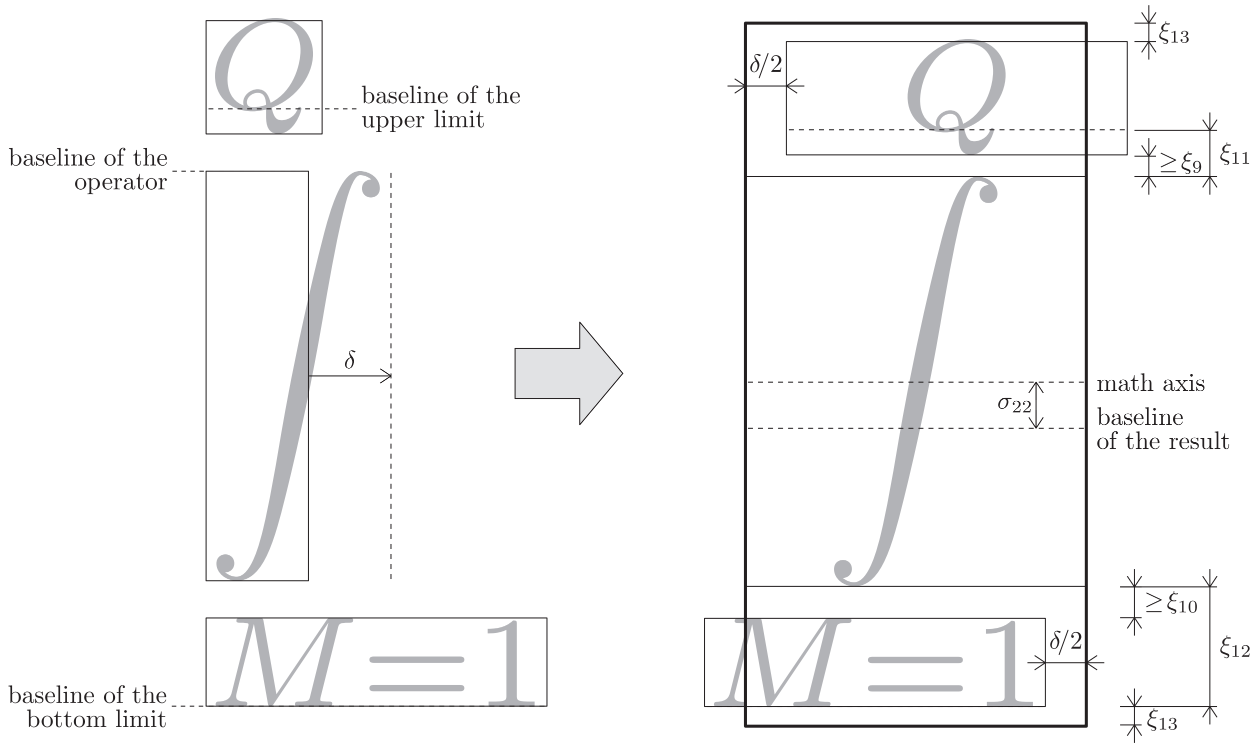


Figure 5: Assembling an operator with limits placed above and below; δ denotes the italic correction of the operator symbol

Typesetting operators with limits

The placement of the limits of an operator depends on the current style and the usage of `\limits` and `\nolimits` commands. If the style is D or D' and the operator `\nolimits` was not applied, the limits are placed above and below the operator, as displayed in Figure 5; otherwise, unless the operator `\limits` was used, the limits are processed as fractions (see following section about fractions).

The operator symbol is centered vertically with respect to the math axis (σ_{22}). TEX tries to place the upper formula in such a way that its baseline is distant by ξ_{11} from the top of the operator; however, if the distance between the bottom of the upper subformula and the top of the operator would be less than ξ_9, the distance ξ_9 is forced. Similarly, the baseline of the lower subformula is distant by ξ_{12} from the bottom of the operator, unless the distance between the top of the lower subformula and the bottom of the operator would be less than ξ_{10}, in which case the distance ξ_{10} is forced.

For the correction of the horizontal placement of the limits, the value of the italic correction of the operator symbol (denoted by δ in Figure 5), is used.

Typesetting generalized fractions

There are two kinds of fractions implemented in TEX: with or without a bar between the numerator and denominator. They are typeset using different rules, as shows Figure 6. These rules, however, do not suffice, as the numerator and denominator are likely to collide. TEX cleverly avoids collisions, as is shown in Figures 7 and 8.

For a fraction with a bar, the numerator and denominator are shifted independently in order to provide a minimal gap, φ, between the formulas and the bar. The position of the bar remains intact — it coincides with the math axis (see Figure 7). For a fraction without a bar, a different strategy is used to avoid the collision, namely, both the numerator and denominator are shifted apart so that the gap between them is equal to φ (see Figure 8). Note that φ has a different meaning in the two cases.

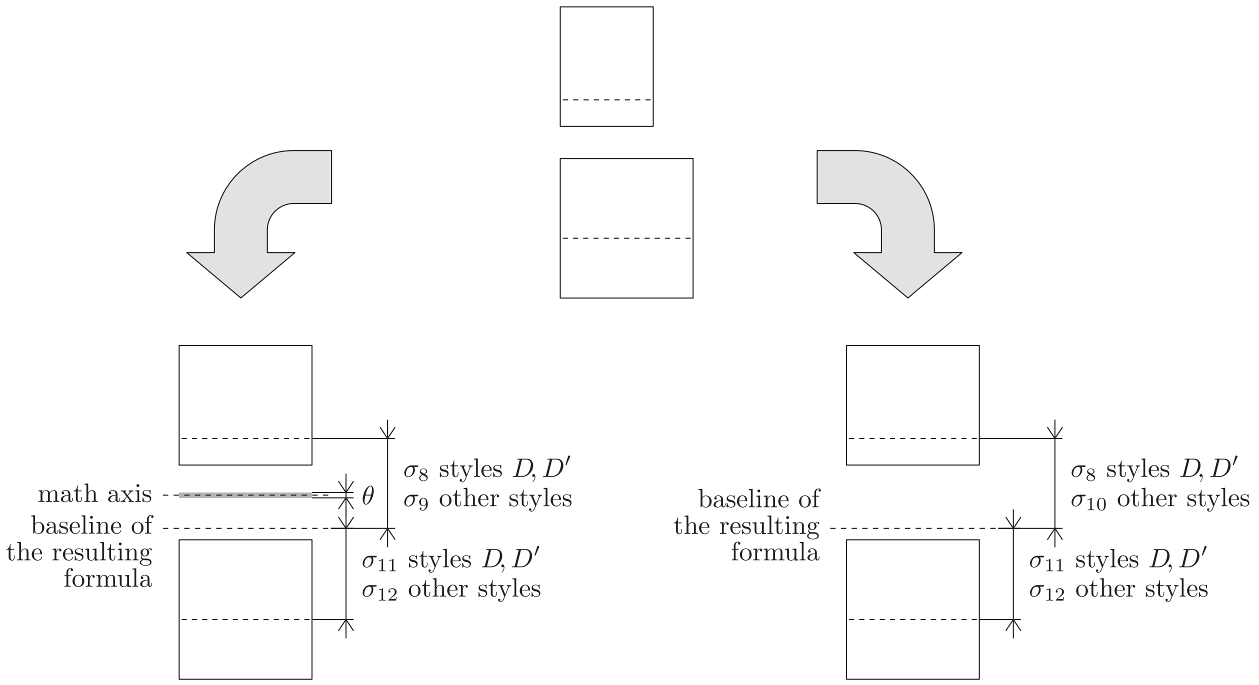

math axis
baseline of the resulting formula
θ
σ_8 styles D, D'
σ_9 other styles
σ_{11} styles D, D'
σ_{12} other styles
baseline of the resulting formula
σ_8 styles D, D'
σ_{10} other styles
σ_{11} styles D, D'
σ_{12} other styles

Figure 6: The placement of numerators and denominators in generalized fractions; the thickness of the rule, θ, is given either by the value of ξ_8 or explicitly; the latter possibility is provided by the `\abovewithdelims` command; observe that σ_9 is used for the formula with a bar, while σ_{10} for the formula without a bar

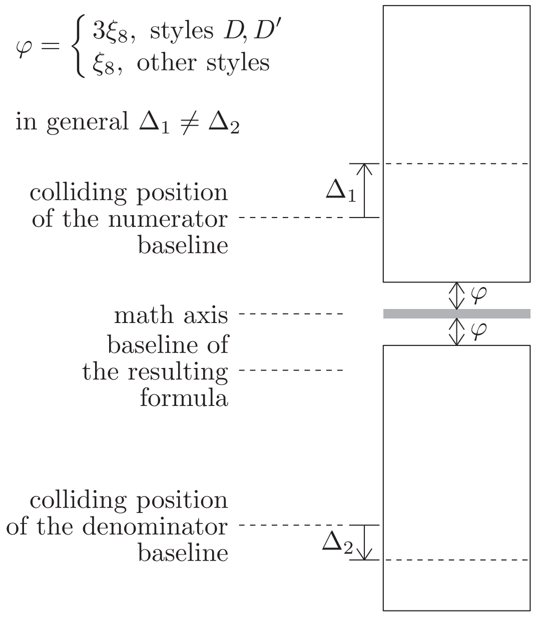

$\varphi = \begin{cases} 3\xi_8, & \text{styles } D, D' \\ \xi_8, & \text{other styles} \end{cases}$
in general $\Delta_1 \neq \Delta_2$
colliding position of the numerator baseline
Δ_1
φ
math axis
φ
baseline of the resulting formula
colliding position of the denominator baseline
Δ_2

Figure 7: Resolving a collision between the numerator and denominator in the case of a fraction with a bar

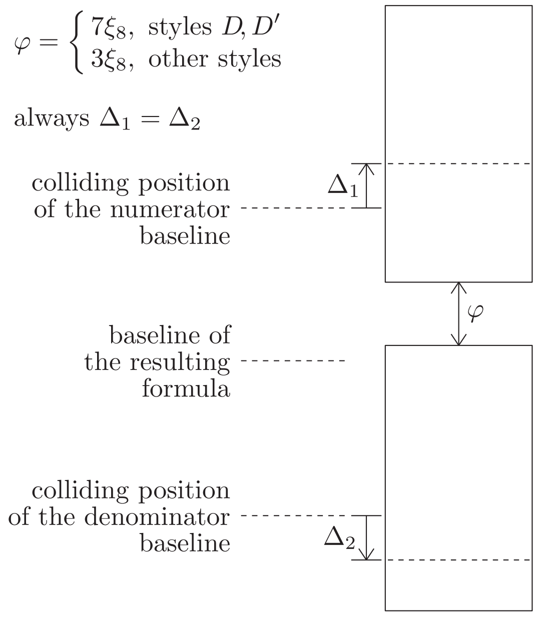

$\varphi = \begin{cases} 7\xi_8, & \text{styles } D, D' \\ 3\xi_8, & \text{other styles} \end{cases}$
always $\Delta_1 = \Delta_2$
colliding position of the numerator baseline
Δ_1
φ
baseline of the resulting formula
colliding position of the denominator baseline
Δ_2

Figure 8: Resolving a collision between the numerator and denominator in the case of a fraction without a bar

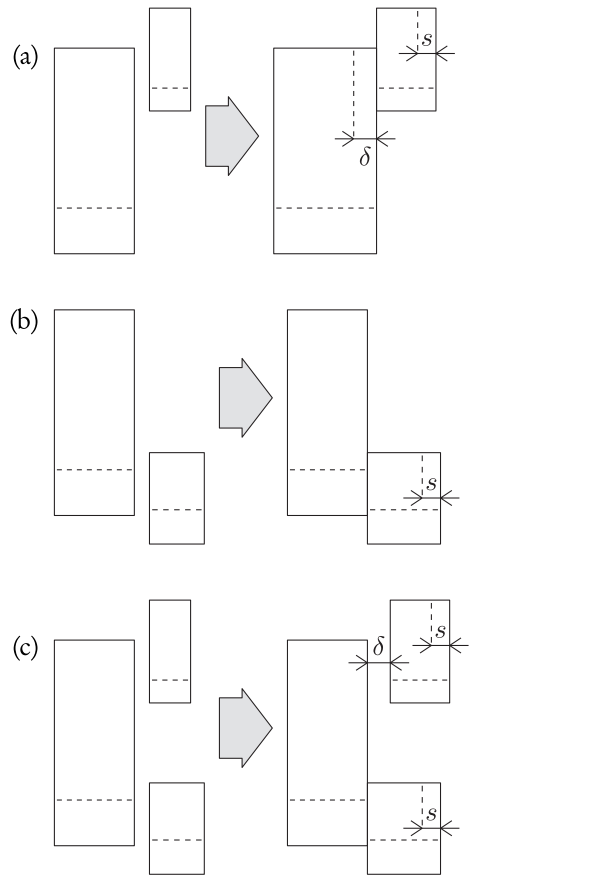


Figure 9: The horizontal placement of indices: (a) the placement of a lone superscript, (b) the placement of a lone subscript, (c) the placement of both superscript and subscript

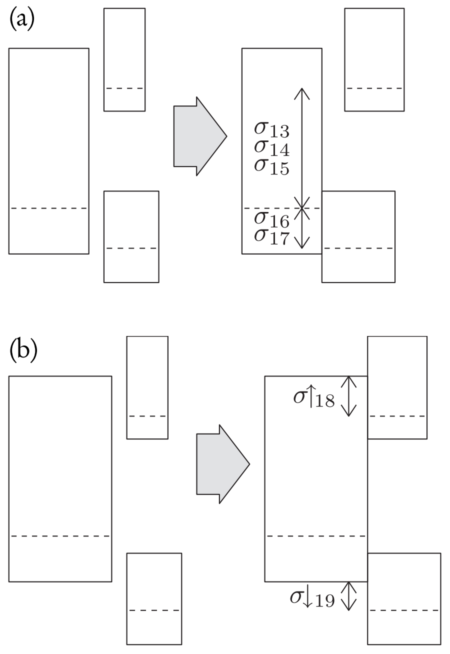


Figure 10: The vertical placement of indices: (a) the placement when the kernel is a symbol, (b) the placement when the kernel is a boxed formula

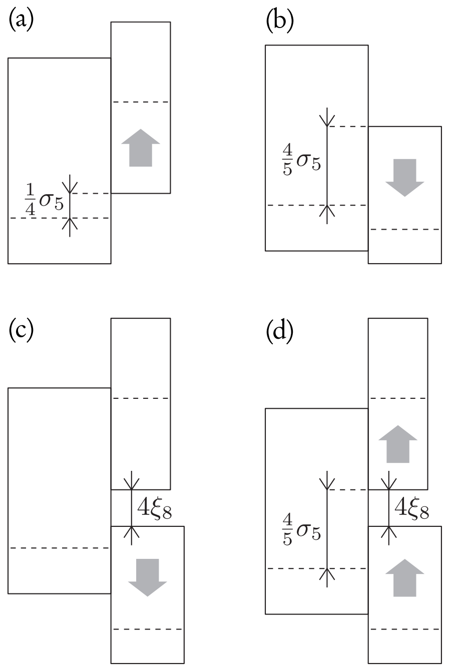


Figure 11: Resolving collisions of indices (further explanations in the text)

Typesetting formulas with indices

The placement of indices is a fairly complex task due to the variety of situations that may occur.

Figures 9a – 9c show the horizontal placement of indices. If the kernel is a single symbol, the superscript, if present, is shifted to the right by the amount of the italic correction of the kernel symbol. Technically, a slightly different procedure is involved in the presence of a subscript, as stated in the description of step 17 in *Appendix G*. If the kernel is already a boxed formula, TEX assumes that $\delta = 0$. A kern of the value `\scriptspace` (*s* in the figure) is always appended to index formulas.

The procedure for the vertical placement (see Figures 10a – 10b) makes use of 7 parameters: from σ_{13} to σ_{19}. Again, different procedures are employed depending on the structure of the kernel. If it is a symbol, σ_{13} for the style *D*, σ_{14} for other uncramped styles, and σ_{15} for cramped styles are used for the placement of a superscript; for the placement of a subscript, σ_{16} is used if a superscript is absent and σ_{17} otherwise. If the kernel is a boxed formula, σ_{18} is used for the positioning of the superscript and σ_{19} for the positioning of the subscript. Moreover, the respective values are not taken from the current font: $\sigma\uparrow$ and $\sigma\downarrow$ mean that the parameters refer to the fonts corresponding to the styles $C\uparrow$ and $C\downarrow$, respectively.

Indices, as one can expect, are also subject to potential collisions. The actions for such a case are depicted in Figures 11a – 11d: (a) the bottom of the superscript formula cannot be placed lower than $\frac{1}{4}\sigma_5$ above the baseline; (b) the top of the subscript formula cannot be placed higher than $\frac{4}{5}\sigma_5$ above the baseline; (c) the gap between the bottom of the superscript and the top of the subscript cannot be smaller than $4\xi_8$ (recall that ξ_8 stores the math rule thickness) — the subscript is shifted if required; (d) finally, if the latter situation occurs, both indices can be shifted up so that the bottom of the superscript is not lower than $\frac{4}{5}\sigma_5$ above the baseline.

Conclusions

As originally mentioned, I prepared the illustrations initially for myself and only later decided to publish them in hope that somebody else may also benefit. Therefore, I eagerly welcome any feedback.

Acknowledgements

I am very grateful to Jerzy Ludwichowski and Piotr Strzelczyk for their prompt and willing help.

References

[1] Donald E. Knuth, *The TEXbook*, Computers & Typesetting: Volume A, Addison Wesley, 1986

[2] Davide P. Cervone, *jsMath: A Method of Including Mathematics in Web Pages*, `http://www.math.union.edu/ dpvc/jsMath/`

[3] Murray Sargent III, *Unicode Nearly Plain-Text Encoding of Mathematics*, `http://www.unicode.org/notes/tn28/`

Open-belly surgery in Ω_2

YANNIS HARALAMBOUS, GÁBOR BELLA, ATIF GULZAR
ENST Bretagne, CS 83 818, 29 283 Brest Cedex 3, France
`yannis dot haralambous (at) enst-bretagne dot fr, gabor dot bella (at) enst-bretagne dot fr, atif dot gulzar (at) gmail dot com`

Abstract

TEX and its successors, including the initial version of Ω, all suffer from the same technical limitations, such as inadequate support for TrueType/OpenType font formats and the lack of distinction between character and glyph data. In this paper, the authors present Ω_2, which provides extensibility through both external modules *and the* texteme concept *that supersedes TEX's tokens and nodes as well as characters and glyphs. Ω_2's modules, while much more powerful than macros or ΩTPs, provide relatively easy access to Ω_2's internals without needing to touch the source code itself. Among immediate applications are full OpenType support (GSUB, GPOS, etc.), use of independent linguistic tools such as hyphenation algorithms, and support for Unicode's Bidirectional Algorithm.*

Introduction

Since its birth, TEX has undergone significant evolution, resulting in extended versions such as ε-TEX, pdfTEX, Ω_1, and others. However, the fundamentals of TEX have barely changed: to cite two examples, both its basic text model, that is, the *horizontal node list*, and the concept of the *single main vertical list* are almost exactly the same as 25 years ago. Ω_1, as a first step towards Unicode compatibility, introduced 16-bit character codes and some text directionality support but did not change TEX's original text model, based on token lists converted to node lists and finally to DVI instructions.

Users and developers have long since recognised the serious limitations of this approach. Inside the belly of TEX, character codes included in tokens are replaced by glyph codes, resulting in loss of information if one does not stick to the severely limited TEX font encodings, especially in the case of non-Latin scripts: searchability and recovery of the original character data in general become impossible. Support for advanced font formats such as OpenType, essential for writing systems having contextual properties (Arabic, Hebrew, Nastaleeq, the Indic scripts, etc.) but also necessary for some Western typographical features, is also impossible without a clear distinction between the concepts of character and glyph. Still due to the same limitation, until now, no successor of TEX could get rid of the TFM font format and provide native support for PostScript or TrueType-based fonts.

As of today, the most remarkable development in the TEX world regarding support for intelligent font formats is without doubt the XƎTEX system [1]. However, as far as micro-typography is concerned, XƎTEX does not have much to do with Knuth's original TEX: while the latter is a stand-alone tool, XƎTEX 'outsources' all the word-level typography to the underlying operating system and external libraries: the ICU library initially developed by IBM [7], ATSUI under Mac OS X, FreeType under Linux, all come into play. So, while XƎTEX succeeds in combining the OpenType and Apple AAT font technologies with TEX's layout and input style, it ties the application to the operating system.

The main reason for preferring such a solution was without any doubt the opportunity to avoid reimplementing the quite complicated Unicode and OpenType engines that already exist on the operating system level. Moreover, the TEX source code itself is far from being easily extendable, despite having been written in the didactic WEB programming language that divides the code into small, easy-to-digest chunks. It lacks modularity and is so highly optimised for performance that the slightest modification can cause a snowball effect of patching and debugging. The single way of extending TEX foreseen by Knuth was the creation of new *whatsit* node types, each of which in practice results in a further increase of the programme's complexity.

Unlike XƎTEX's approach, the developers of Ω_2 preferred to solve the problem of extensibility by introducing modularity into the system. In fact, ΩTPs were already module-like components in Ω_1. ΩTPs are capable of transforming 'character' strings into other 'character' strings and even of inserting new control sequences. However, due to the early, token-level stage where they intervene, they are limited by the grouping of input text: to show an example, processing of the word '*emph*asis':

```
{\it emph}asis
```

as a whole is not possible, since it is broken by markup into two separate ΩTP buffers. Even more important is the fact that ΩTPs are inherently character-level tools and are unable to perform operations other than character substitutions (such as glyph positioning, adding linguistic data, modifying colour, etc.).

Consequently, the objectives that the Omega team have set to themselves are on one hand to solve the character/glyph duality issue, by creating data structures capable of storing both, and on the other hand to provide extensibility to Ω_2 in a more efficient way, in order to allow manipulation of characters, glyphs, and other types of data independently. It will be shown later in the article how these two improvements are tightly related and that they provide the best results when used together.

In the following section, the original text model of TEX will be compared to our *texteme*-based approach. Then, external modules will be presented in detail. Finally, it will be shown how using modules together with texteme properties opens up possibilities of immediate applications such as OpenType support, linguistic analysis, or fully customised typography beyond the limitations of TEX or current font technologies.

Of characters, glyphs, tokens, nodes, and DVI instructions

Text in TEX and in its extensions goes through several different states. In the beginning, it is read as *character data* from the input buffer. These characters may either be textual content or TEX markup. They will immediately be converted into either *character tokens* or *control sequence tokens*, respectively. A character token consists of the character code and its catcode, while a control sequence is represented by a single identifier. However, the token state is ephemeral: shortly after their creation, both types of tokens are converted into *nodes*.[1] Character tokens usually become character nodes, but sometimes also ligature nodes. Other types of nodes are also created on-the-fly: kern nodes, glue nodes, discretionary nodes, and so on. Text is organised into horizontal and vertical lists (represented by *hlist* and *vlist* nodes).

An important thing to notice is that fonts come into the picture precisely at the point of converting tokens into nodes. (Ligature, kerning, glue, etc., information all come from font resources.) This is the very moment of TEX's original sin: supposing that

character code ≡ *glyph code from the font*,

characters are being *replaced* by glyphs in character nodes. The reason why hyphenation, in principle a character-based operation, still works at a latter stage[2] is this assumed equality, that is in fact valid only for a small set of characters, namely those that were coded in locations common between character and font encodings. Were we to use a different (say, OpenType) font format or a script like Arabic, subsequent character-based operations such as searching, copying and pasting, and hyphenation would all be doomed to failure.

The odyssey is still not over: TEX, having done most of its work on node lists, in the end outputs the resulting document using the venerable DVI format where text is encoded through glyph identifiers only, represented in 16 or 8 bits, depending on whether Ω_1 or another TEX-based system is being used. By this time, all the other types of nodes holding non-character data either have already been absorbed in the typesetting process (penalty, discretionary, etc.), or else they now become physical dimensions (kerning, offsets, glue, etc.) or special DVI instructions (specials, etc.). This is the end of the story: our output DVI document is purely presentation-oriented and in no way is able to provide the original character data, long lost in the process.

Textemes

Textemes[3] are one solution to the problems presented above. The idea is to replace TEX's various data repre-

[1] ΩTPs extend the lives of tokens somewhat: they read character tokens and output both character and control sequence tokens.

[2] Namely, at line breaking.

[3] Introduced as *signs* at the EuroTEX 2005 conference.

sentations, namely characters, character tokens, some types of nodes, as well as glyphs in the output, by a single entity: the texteme.

A texteme, as presented in detail in [5] and in [4], is a *set of properties*, where a property is basically a *key–value pair*. A texteme usually represents a character, its glyph, and other related data. More specifically, character code, glyph and font identifiers, and any kind of information related to the atomic units of electronic text are all represented by texteme properties.

How do textemes work in Ω_2? The general idea is to let information accumulate inside textemes instead of converting data from one form to the other. Raw, unformatted text is a texteme string where textemes contain only *character* properties. When raw text (with markup) is fed into Ω_2, a *catcode* property is added to every texteme. When font information is read, instead of creating character nodes, the same textemes — texteme nodes — are carried on, only with new *glyph* and *font* properties added. No ligature nodes are created: an 'fi' ligature is represented by two textemes linked together, the first with *character*=`f` and *glyph*=fi, and the second with *character*=`i` and *glyph*=∅ (empty). Some other information like kerning, glue, or penalties, become texteme properties just the same, resulting in simplified text structure.

Separating character and glyph codes while having access to both of them throughout the whole typesetting process proves to be very useful for tasks such as hyphenation (as shown in Haralambous's article [3]). However, the ultimate goal is to be able to produce final documents that keep all these accumulated (and useful) information. A document that displays glyphs but also holds the original character-based text has an enormous technical advantage compared to glyph-only documents where retrieval of characters is only possible through non-standard, error-prone glyph naming schemes.

The PDF format makes such a double encoding of text possible through the `ActualText` operator: for every glyph or glyph sequence, the corresponding character or character sequence may be defined. This way, even cases like multiple glyphs corresponding to a single character or reordered glyph sequences can be handled correctly, something that would never be possible through glyph naming.

Unfortunately, Ω_2 does not (yet) produce PDF directly and the DVI format does not offer mechanisms similar to PDF's `ActualText`. It is therefore not possible to output texteme data into DVI without breaking compatibility with the original DVI format. As a temporary solution, Ω_2's DVI format has been slightly modified in order to include texteme-related information that is interpreted by a patched `dvipdfmx` utility that produces PDF documents with `ActualText` operators. This is a quick and dirty solution, but it works.

The document creator is by all means allowed and encouraged to invent and use their own properties in their documents. First of all, the set of available texteme properties is open and extensible.[4] Such user-defined properties can be added either automatically, by linguistic analysers and various text processor tools, or manually, by a texteme-compatible text editor (a simple prototype of which has been developed by students of ENST Bretagne). In this editor, texteme properties are added and manipulated in an intuitive, graphical way. Texteme-based documents can then be saved in XML that Ω_2 will be able to interpret and thus rebuild texteme strings. (The XML reading capability of Ω_2 has not been developed yet.)

How do texteme properties come into play during text processing? External modules are the answer.

External modules

Theoretical considerations

As mentioned before, Ω_1's ΩTPs are basically character processors at the token level. This approach is not sufficient when non-character data needs to be processed (e.g., glyph substitution or glyph positioning). First, with the introduction of textemes, access to individual texteme properties as opposed to mere character strings becomes necessary. Secondly, even if an extended ΩTP syntax and input/output scheme allowed the handling of such information, ΩTPs are still called at the token level where font data have not yet been read by Ω_2.

Consequently, in order to allow ΩTP-like external modules to process font-dependent information, their point of activation needs to be displaced to a later point, to the node level.

But there is a problem: since nodes (including texteme nodes) and node lists are considerably more complicated data structures than characters or tokens, Ω_1's internal ΩTP approach is not powerful enough to de-

[4]Namespaces are used for semantic disambiguation.

scribe transformations on them. For these reasons, our new external modules need to be standalone binaries that communicate with Ω_2 using a well-defined XML format (more on this later). Since these binaries can be written in any programming language, there is no limitation to their computing power, unlike former internal ΩTPs that were equivalent to finite state automata.

In reality, there are no less than three well-defined points during Ω_2's typesetting process where external modules may be called:

1. on token-based input text (as in the case of ΩTPs);
2. on yet unbroken horizontal node lists that represent whole paragraphs;
3. on node lists representing individual lines during paragraph breaking.

Each of these three legal intervention points corresponds to a set of well-defined processing tasks. The first point is used by character-level transformers and analysers. They receive a simple list of textemes, uninterrupted by control sequences on grouping braces (no wonder: these tokens act as boundaries for the ΩTP buffer), and containing mostly character information. They are supposed either to perform character transformations (e.g., converting from a local transcription scheme or encoding to Unicode, preprocessing, etc.) or to generate new texteme properties. (At the moment of writing the article, Ω_2 is not yet capable of adding texteme properties at this stage.)

The second type of external module reads entire paragraphs and operates on node lists. This type of module applies, among others, OpenType glyph substitution and positioning rules. However, as a result of working with nodes instead of just characters, such modules have enormous power as well as responsibility over the behaviour of Ω_2: they have full access to every aspect of the text including horizontal and vertical lists, kerning, penalties, and so on. Were a module to, say, substitute a glyph by another, it would have the responsibility to update the corresponding kerning information or at least make sure that this will be done subsequently either by Ω_2 or by another module.

Finally, the third type of module is called on individual lines, inside the line breaking algorithm. Similarly to modules of the second type, it operates on node lists. Its task is to perform line-related operations such as optical kerning, OpenType JSTF (justification) support, or line-dependent glyph substitutions and positionings (e.g., an OpenType contextual ligature invalidated by a nearby line break).

The fragility of node lists when manipulated externally may seem worrying. Indeed, it is very easy to produce typographically unacceptable documents and even to freeze Ω_2 through erroneous or malicious node operations. Creators of modules should respect rules regarding what and in what order they are allowed to modify. Correct ordering of modules is of crucial importance: for example, the order *character transformations – glyph substitutions – glyph positionings* should always be respected, otherwise regression problems may arise.

Indeed, node-level modules represent a drastic surgical intervention in Ω_2's digestive system: it is as if Ω_2's stomach and intestines were piped into external digestion machines. The reader will kindly excuse the authors for this somewhat disturbing analogy and read on to see how in practice modules are called from Ω_2.

Modules in Practice

Module support in Ω_2 is currently in prototype stage, that is, developer- and user-friendly macros and libraries are only minimally available at the moment.

External modules are implemented as standalone binaries and communicate with Ω_2 through signals and an input-output buffer. For performance reasons, these binaries run as *daemons*, that is, they are spawned only once and then remain idle until they receive data to process as well as a wakeup signal. Once a module has finished processing, after writing its output into the same input-output buffer, it goes back to sleep again, and Ω_2 continues by waking up the following module.

More precisely: a module binary is launched by the `\registermodule` primitive. By writing

```
\registermodule{mymodule}{modbin}{par}{10}
```

the binary programme `modbin` is run, in the future referenced by the name `mymodule`. The `par` parameter means that it is a type 2 (paragraph-level) module and its number in the execution order among modules of the same type is `10`. These four parameters are mandatory. There is a fifth, optional parameter (omitted in our example) that sends its argument to the binary when it is launched; this may sometimes be useful for initialisation purposes.

Modules are *asleep* by default. For performance reasons, Ω_2 does not send any data to sleeping modules. In order to perform their task, modules need to be both *waked up* and *activated*. They are waked up and sent back to sleep by the `\wakeup{mymodule}` and `\gotosleep{mymodule}` primitives. Note that paragraph- and line-level modules are waked up for *entire paragraphs*, there is no point in trying to wake them up for shorter text segments. *Awake* but *inactive* modules receive and read all data but let them pass through untouched. They activate themselves when they encounter an *activate* special node, inserted by the `\activate{mymodule}` macro. From this point, they process the text until they either read a *deactivate* node or arrive at the end of the buffer. These *activate* and *deactivate* nodes may of course very well appear inside paragraphs, making it possible to activate modules for text segments as small as individual characters (more precisely, textemes).

Text (textemes and other nodes) is transmitted between Ω_2 and modules in XML format. The full DTD is not provided here for space reasons; instead, a small but relevant example is given. As is shown, for paragraph and line-level modules, Ω_2's *current font table* is also included in the XML buffer since these modules (e.g., an OpenType engine) usually need access to fonts. The font table is then followed by the node list itself: in our case, a *whatsit*, an empty *horizontal list* and two *texteme* nodes. In this simple example, texteme nodes contain only three properties each, namely the character and the corresponding font and glyph ID.

```
<?xml version="1.0"?>
<buffer version="0.1">
  <preamble>
    <fontlist>
      <font id="51" name="ptmr"
            size="1310720"/>
      <font id="52" name="pala"
            size="655360"/>
      ...
    </fontlist>
  </preamble>
  <nodelist>
    <!-- whatsit -->
    <wha st="6" intpnl="0" brkpnl="0"
         pardir="0">
      <lbl></lbl><lbr></lbr>
    </wha>
    <!-- hlist -->
    <hls wd="1310720" dp="0" ht="0"
         shift_amount="0" gse="0" gsi="0"
         go="0" dir="0">
    </hls>
    <!-- texteme -->
    <t linkl="0" linkr="0">
      <p n="c">76</p> <!-- char: L -->
      <p n="g">12</p> <!-- glyph -->
      <p n="f">52</p> <!-- font -->
    </t>
    <t linkl="0" linkr="0">
      <p n="c">111</p> <!-- char: o -->
      <p n="g">142</p>
      <p n="f">52</p>
    </t>
    ...
  </nodelist>
</buffer>
```

A particular advantage of communicating with modules in XML is that certain tasks can thus be implemented by very simple XSLT transformations that are executed by a generic XSLT driver module. This way, the task of implementing a module is reduced to the complexity of writing XSLT code.

Applications of modules and textemes

OpenType support

For quite a long time, the Holy Grail of typesetting systems has been to implement robust support for the TrueType and OpenType font formats. Development has been slow on all platforms, due to the investment required on a very wide scale (full Unicode support, internationalisation, availability of actual fonts). No wonder that no TEX-based system, apart from XƎTEX, has even come close yet to full OpenType compatibility: without the profound changes in TEX's text model described in earlier sections of the present article, or at least similar improvements, OpenType support is not fully possible.

As has been shown in numerous articles, such as [1], the main difficulty of developing OpenType-compatible applications lies in the complexity of operations described in OpenType's GSUB (glyph substitution) and GPOS (glyph positioning) tables. In order to achieve this, apart from providing an appropriate text model, typesetting applications also need a powerful OpenType engine. Fortunately, a lot of effort has already been made in this direction in the free software community, and thus free and cross-platform OpenType libraries are already available: let us mention the *FreeType* [6] and *M17N* [2] projects that both offer OpenType-related functionalities. The new Ω_2 system makes use of both of these libraries.[5]

Basically, two aspects of the OpenType format need to be taken care of in Ω_2: reading metrics and performing glyph transformations. Most TEX-based systems solve the former issue by falling back to utilities such as `ttf2afm` that convert TrueType metrics into TFM files, the OpenType and the TrueType formats being compatible as far as metrics are concerned. This solution works but is inelegant from the user's point of view since installation and use of TrueType or OpenType fonts require several conversion steps and editing of various configuration files. The authors have thus decided to implement direct access from Ω_2 to OpenType metrics, without any need for OFM or other files. At the moment of writing the article, Ω_2 is already capable of reading TrueType metrics directly from the font.[6]

Glyph transformations, on the other hand, are too big a task to implement inside the monolithic Ω_2 code. Modules are an ideal place for such operations. Both paragraph- and line-level modules are going to be necessary: paragraph-level modules will perform both font-independent (multilingual preprocessing similar to what Microsoft's Uniscribe library does) as well as font-dependent OpenType GSUB and GPOS glyph transformations. Finally, the same OpenType module intervenes once again at the line breaking phase if necessary; also, JSTF support can be implemented at this point.

Hyphenation

TEX's original hyphenation procedure is called in the line breaking phase: at this point, text is converted into lowercase, ligatures are replaced by their original characters, and pattern matching is performed. Consequently, the hyphenation algorithm is an integral part of TEX that is difficult to modify or customise according to the special needs of different languages, apart from language-dependent pattern sets. In Ω_2, textemes and modules allow performing hyphenation externally, in a module, opening up the possibility of using more advanced hyphenation algorithms. The general idea is that the external hyphenation module marks potential hyphenation points in words using texteme properties and at the line breaking stage Ω_2 simply selects from the marked hyphenation points the ones giving the least badness.

See [3] (in this same proceedings volume) for a more detailed description of new hyphenation techniques used in Ω_2.

Getting rid of (some) TEX markup

Through properties, textemes provide a new way of enriching electronic text. In some cases, such properties can substitute for markup that would otherwise serve the same purpose. The interest in doing so lies in the simplification of input text, an important gain both from a technical and a usability point of view. Consider the following example of LATEX code:

```
The \verb#\textcolor# command's purpose
is to colorify text, such as this word
in \textcolor{red}{red}.
```

There are several problems with the above snippet: first of all, it is far from being intuitive. To typeset the '\' character, the user needs to use the `\verb` command, which is one way of escaping the special role of the backslash. Also, there is nothing indicating that the first parameter of the `\textcolor` command is the colour parameter and the second is the text to be coloured: neither a human nor an automatic tool, say a preprocessor, can distinguish them without proper knowledge of the `color` LATEX package. Finally, the use of control sequences and delimiters breaks up text into small chunks causing various problems at later processing stages.

A possible solution is to use texteme properties for colour as well as for escaping. For example, with a *catcode = 12* property the user could mark the backslash as textual content. Of course, more user-friendly property name and values could also be used. The same point can be made for various types of spaces (non-breakable, thin, etc.): instead of using active characters like '~' or control sequences like '`\,`', such information can be encoded as orthogonal properties of the same space character. This solution is also far simpler and more intuitive than using Unicode's various *non-breakable space*, *thin space*, *non-breakable* and *thin space*, etc., characters.

[5] M17N, far from being just a multilingual typesetting engine, is a whole set of libraries aiming at providing complete multilingual support.

[6] Kerning data is not read by Ω_2, as this operation is planned to be implemented inside an external module.

Linguistic tools

One of the advantages of textemes is that the set of available properties is open which, together with modularity, makes it possible to integrate Ω_2 with new text processing applications. For numerous linguistic problems that bear some relation to typography, such an approach can be fruitful: automatically finding word boundaries in Thai or Chinese text or distinguishing dots (for abbreviations) and periods (at ends of sentences) in English typography are all complicated linguistic problems that transcend the limits of typesetting engines. If appropriate, standalone linguistic tools already exist and are able to perform the tasks in question; they can be called as external modules of Ω_2 in order to add linguistic information to the input text in the form of texteme properties. Then, an Ω_2 developer just needs to implement a second, much simpler module that interprets such linguistic properties and takes them into account at the typesetting stage (correct line breaks for Thai, changes in the widths of spaces for English).

Conclusions

The second version of the Ω_2 system has two new aspects: *texteme support* and *modularity*. Although still in a prototype stage, the basic framework for running external modules has been implemented in Ω_2 and the underlying text model has also been adapted to the texteme concept. As an addition, the new Ω_2 outputs a DVI format that contains both characters and glyphs, and with a patched `dvipdfmx` utility this information can be incorporated into PDF documents that as a result will be able to provide the reader both with formatted output and with the *original* character string. On the input side, a texteme-compliant text editor tool has been developed, allowing users to enter texteme properties into input documents. OpenType support is partially available: Ω_2 now reads metrics from OpenType files directly. Work is underway for modular GSUB and GPOS support. The authors are also working on moving TEX's hyphenation algorithm into an external module, resulting in easier implementation of improved hyphenation tools.

Work that still needs to be done includes full capabilities of texteme input and output: texteme-based auxiliary (`.toc` etc.) files and input format, as well as development of various multilingual modules including support for OpenType layout tables. An especially important feature that still needs further development is generation of PDF documents with both character and glyph information added. The already mentioned `dvipdfmx` tool is a likely candidate for such an extension. The authors kindly welcome contribution from the TEX community in these areas.

References

[1] Jonathan Kew: *The Multilingual Lion: TEX Learns to Speak Unicode*. 27th International Unicode Conference. *TUGboat* 26:2, 2005, pp. 115–124.

[2] Nishikimi Mikiko, Handa Kenichi, Takahashi Naoto, Tomura Satoru: *The m17n Library — A General Purpose Multilingual Library for Unix/Linux Applications*. Asian Symposium on Natural Language Processing to Overcome Language Barriers (2004). `http://www.m17n.org`

[3] Yannis Haralambous: *New Hyphenation Techniques in Omega 2*. Proceedings of the EuroTEX 2006 (Debrecen, Hungary) conference. In this volume, pp. 98–103.

[4] Yannis Haralambous, Gábor Bella: *Injecting Information into Atomic Units of Text*. ACM Symposium on Document Engineering 2005.

[5] Yannis Haralambous, Gábor Bella: *Omega Becomes a Sign Processor*. Proceedings of the EuroTEX 2005 (Pont-à-Mousson, France) conference, pp. 99–110.

[6] *The FreeType Project*. `http://www.freetype.org`

[7] *International Components for Unicode*. `http://icu.sourceforge.net/`

New hyphenation techniques in Ω_2

YANNIS HARALAMBOUS
Département Informatique, ENST Bretagne, CS 83 818, 29 283 Brest Cedex 3, France
`yannis dot haralambous (at) enst dash bretagne dot fr`

Abstract

By replacing the internal hyphenation engine of TEX by an external Omega$_2$ module, we are able to solve all shortcomings related to hyphenation and to add new features: segmentation of compound words, excentricity, preferential hyphenation.

Introduction

Ever since a computer hyphenated the word "God" and ruined a night's sleep of an RCA employee, there has been quite a lot of literature on hyphenation:[1] it is a complex linguistic operation, permanently in use (at least for languages that are hyphenated), and requiring high efficiency. Nevertheless there are some flaws in TEX's approach to hyphenation, as well as some areas where extra features could be helpful.

The flaws are mainly (a) the fact that hyphenation patterns are stored in the format, so that one needs to know in advance which languages will be used in the document and create the appropriate format file; (b) in some contexts, words are not hyphenatable because they are not preceded by glue (for example, a word preceded by a penalty, or the first word of a paragraph); (c) font or color changes prohibit hyphenation, so that a word like "différ*a*nce" cannot be hyphenated; and (d) special hyphenations such as German *backen* becoming *bak-ken* are possible (through the *discretionary* primitive) but cannot be automatic.

New features have been suggested on many occasions: for example, it would be very useful for some languages to prioritize hyphenation between word components rather than between syllables in the same component. In German, the priority list is even threefold: first comes hyphenation between components, then hyphenation inside the last component, and last *and* least: hyphenation inside the other components. Another useful extra feature is weighted hyphenation: for example, in French, words starting with "con" should not be hyphenated at that location, but this restriction should not be absolute: if one cannot do otherwise, it should be allowed to hyphenate nevertheless (a typical example is the word "conscience" which can be hyphenated only after "con"). So one should be able to specify a penalty value for each hyphen. Another useful feature would be interaction with the user in case of ambiguity requiring morphological, syntactical or even semantic input: a typical case is already stated in *The TEXbook*: "rec-ord" (the noun) vs. "re-cord" (the verb). Whenever the algorithm detects such an ambiguity, the user should be warned and, why not, asked to disambiguate.

In languages like Greek there is no requirement for hyphenating between word components.[2] Nevertheless, although it is allowed, it looks silly to hyphenate a word such as Π*απαχατζηχαραλαμπό- πουλος* after the two first letters. In such cases it would be useful to prioritize hyphenation towards the middle of the word rather than near its borders.

In this paper we present the new hyphenation module of Omega$_2$, which solved the problems mentioned and provides infrastructure for the extra features described above.

[1]See, in particular, [1, 6, 7, 8, 10, 11, 12, 13, 14].

[2]The reader may wonder why there is such a requirement for German and not for Greek, which uses as least as many compound words as German. Here is a possible explanation: in German, compound words are separated by a glottal stop. For example, *Satzende* will be pronounced "zats[break]ende" to distinguish the components *Satz* (= sentence) and *Ende* (= end). The visually similar *Sitzende* will be pronounced continuously "zitsende" (= sitting). In Greek, however, there is no glottal stop: *συνάδελφος* (= colleague) is pronounced continuously "sinathelfos" although it is composed by *συν* (= plus, together) and *αδελφος* (= brother). This feature of the modern Greek language has influenced hyphenation practice. In fact, there are two ways to hyphenate this word in Greek: in modern *dêmotikê* [5] it will be hyphenated phonetically *συ-νά-δελ-φος* (which contradicts component segmentation), and in *kathareuousa* it will be hyphenated etymologically *συν-άδελ-φος*. The question of whether *kathareuousa* hyphenation patterns should give priority to word components is open.

Description

A *module* for Omega$_2$ is a binary reading and writing horizontal node lists serialized in XML. It is hooked into Omega$_2$ at two possible locations: one is inside the *end_graph* procedure (§1096 of [9], just before the call of *line_break*: that's when a complete paragraph is sent to the paragraph builder). The second location (which has not been implemented yet) is inside the paragraph builder, for a given vertice of the graph of break nodes.

To say it simply: a module extracts horizontal node lists from Omega's stomach, modifies them, and puts them back so that typesetting can go on.

But there is something more in Omega$_2$: instead of character nodes, we use *texteme* nodes [3], so that we clearly separate glyphs from characters and that we can add arbitrary name/value pairs to each node.

In our case, the hyphenation module will study the paragraph, apply the hyphenation algorithm to textemes and add a "potential hyphenation point" property to some of them. The value of this property is not simply a boolean but rather a penalty, so that the paragraph builder will automatically prioritize some hyphenation points (for example, those between word components).

Problems solved

Let us see how our approach solves the five problems described in the first section.

Problem a: Preloading of patterns

Omega$_2$ does not preload any patterns in the format file. Patterns contained in external files are dynamically loaded by the module (and subsequently kept in cache), whenever they are needed.

Problem b: Unhyphenatable words

This problem has always been a nightmare for TEX users: now and then, for reasons which seem obscure to the average user, a word will not be hyphenated, resulting in a horrible overfull box. Most of the time the reason is the fact that the word is preceded by something which is not glue. Indeed, according to §891 of [9], *a "potentially hyphenatable part" of a paragraph is a sequence of nodes $p_0p_1\ldots p_m$ where p_0 is a glue node, $p_1\ldots p_{m-1}$ are character, or ligature or whatsit or implicit kern nodes, and p_m is a glue or penalty or insertion or adjust or mark or whatsit or explicit kern node* (see also [4]).

Since now hyphenation is external to Omega, we can define our own rules. One can apply the original TEX rules so that we obtain the same results. Or one can define new rules and obtain potentially better results.

Problem c: Font or color change

This problem comes from the fact that, as often in TEX, the rules for hyphenatable words we described above are not complete. Other restrictions arrive as we read §891: *a whatsit node found after p_1 will be the terminating node p_m* and *all character nodes that do not have the same font as the first character node, will be treated as nonletters*. That means that a *special* primitive or a font change inside a word prohibits hyphenation after them.

In fact, the use of textemes allows us to inject into text properties which will not disable hyphenation. A less typical example is vertical offset: up to now, the TEX logo was not a word, but a graphical construction using glyphs. Using texteme properties to specify the lowering of letter 'E' it will be at last possible to hyphenate the title of the well-known journal *Die TEXnische Komödie*!

As said in the previous section, since we define our own rules, this restriction can very well be abandoned.

Problem d: Special hyphenations

There is no miracle for that: the various special cases have been hard-coded in the module (one could imagine a syntax for including them in the patterns, but the author considers that their extreme rarity does not justify a syntax enhancement).

New features

Penalties

As said in [9] §145, a discretionary node produces either a *hyphen_penalty* or an *ex_hyphen_penalty* depending on its pre-break text. This penalty can be changed by the user, but on a global level only, and certainly not separately for each hyphen point. In Omega$_2$, there are 65,536 hyphenation penalty registers. Patterns can contain a hyphenation register number (the default register being 0). The hyphenation engine will transmit the highest register number value to the paragraph builder through a texteme property.

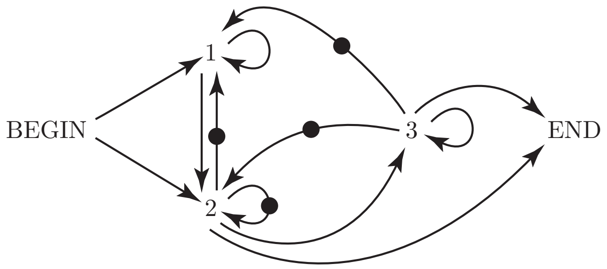


Figure 1: Finite-state engine of *SiSiSi*

The paragraph builder will use the penalty value of that register.

It is up to the user to take advantage of these penalties to prioritize hyphenation between word components, or simply to make the paragraph builder prefer a given hyphenation point rather than some other.

The obvious question which remains is: how do we calculate the various hyphenation penalties in the case of, for example, word component boundaries.

Excentricity

To prioritize hyphenation points that occur near the middle of words, we have introduced a number called *excentricity factor* (a deliberate neologism). This number is the slope of a linear function giving the hyphenation register number according to the distance of an hyphenation point from the center of the word: if ϕ is the factor, i the position of the letter in the word and c the center of the word, then the hyphenation penalty register will be 0 for letter c, $\mathrm{int}((c\cdot i)\cdot s)$ when $c > i$ and $\mathrm{int}((i\cdot c+1)\cdot s)$ otherwise. So, for example, the word

$$\Pi\alpha|\pi\alpha|\chi\alpha|\tau\zeta\eta|\chi\alpha|\rho\underline{\alpha}|\lambda\alpha\mu|\pi\acute{o}|\pi o\upsilon|\lambda o\varsigma$$

with an excentricity factor of, for example, $\phi = 0.33$, will be hyphenated with penalties contained in the following registers:

$$\Pi\alpha|_3\pi\alpha|_3\chi\alpha|_2\tau\zeta\eta|_1\chi\alpha|_0\rho\underline{\alpha}|_0\lambda\alpha\mu|_1\pi\acute{o}|_1\pi o\upsilon|_2\lambda o\varsigma$$

It is up to the user to choose the penalty values for each of registers 0, 1, 2, 3. With a higher slope, we get more registers and are able to control hyphenation penalties more finely.

A Finite-State Engine

There are two ways to calculate word component boundaries. Either by using again patterns (as suggested by Antoš in [1]), or by using a finite-state engine, as used by spelling checkers such as *huspell*.

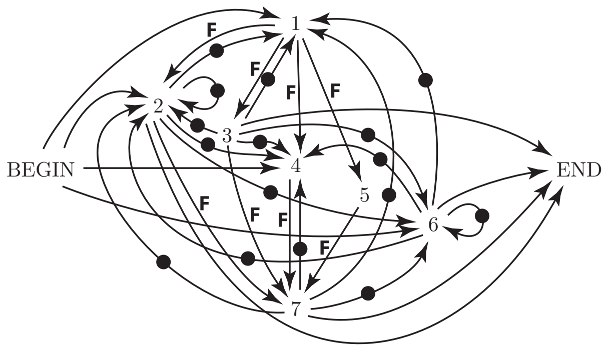


Figure 2: Finite-state engine of *huspell*

Let us develop the second case. A first approach to word component detection through a finite state engine was *SiSiSi* (= *Sichere sinnentsprechende Silbentrennung für die deutsche Sprache* = "Reliable and Sense-Conveying Hyphenation for the German language"), a TEX extension developed in the early nineties by Barth and Steiner [2] based on their work on hyphenation from the eighties. In *SiSiSi* a word is segmented in three parts: a series of prefixes, a single stem and a series of suffixes. This can be described by the finite-state engine in fig. 1: states 1, 2 and 3 are resp. the one of prefix, stem and suffix. We have twelve transitions BEGIN→1 BEGIN→2 1→1 1→2 2→*1 2→*2 2→3 2→END 3→*1 3→*2 3→3 3→END, where the asterisk (or • on the figure) means that going through this transition we enter a new word component.

This approach has been proven to have relatively low efficiency (the *SiSiSi* system was strongly relying on interaction with the user to find and store exceptions to rules).

Another word segmentation approach is the one of spelling checkers. Ispell-based checkers (like *huspell*, which seems to be the most advanced variant) use a "file of affixes" containing lines of the like:

```
SFX A   lig   elig   [^aeiouhlräüö]lig
```

which means: there is a set of rules called A containing this rule; this rule says: when you see a word ending by some string matching the regular expression `/[^aeiouhlräüö]lig/` then strip `lig` and add `elig`. These rules generate new word forms from the existing ones, whenever the latter satisfy the requirements. Similar rules exist for prefixes (starting with PFX).

We have modeled this approach as a finite-state machine with 7 states and 32 transitions. The 7 states are: (1) prefix, (2) stems which are not modified by an

SFX or PFX rule, (3) stems which have been modified by a PFX rule (after modification), (4) stems which have been modified by a SFX rule (after modification), (5) stems which have been modified by both a PFX and a SFX rule (after modification), (6) stem which cannot be combined with either a prefix or a suffix, (7) suffix. It should be noted that only 1, 2, 4 and 6 can be at word begin, and only 2, 3, 6 and 7 can be at word end. The reader can find a graphical representation of this engine in fig. 2.

Here are the transitions: BEGIN→1 BEGIN→2 BEGIN→4 BEGIN→6 1→2F 1→3F 1→4F 1→5F 2→*1 2→*2 2→*4 2→*6 2→7F 2→END 3→*1 3→*2 3→*4 3→*6 3→7F 3→END 4→7F 5→7F 6→*1 6→*2 6→*4 6→*6 6→END 7→*1 7→*2 7→*4 7→*6 7→END. Those marked by an 'F' are conditional transitions: they only apply whenever left and right string belong to the same "family." Families are necessary because of the regular expressions in SFX and PFX rules.

We will see in the next section how this information is included in the patterns file.

Patterns file

TeX users are used to patterns files containing a *patterns* primitive and, in many cases, a *hyphenation* primitive for exceptions. These files sometimes also contain *lccode* commands because only characters with nonzero *lccode* are recognized by TeX's hyphenation algorithm. This part of the code works also as a mapper to equivalence classes, so that patterns can be written using those classes rather than explicit characters. For example, in the case of Greek one can map all combinations of letter *alpha* and diacritics to a single equivalence class and use that class in the patterns. That way, one has fewer patterns and the result is the same (provided, of course, that hyphenation rules are independent of diacritics).

In our new hyphenation patterns files we keep the same pseudo-commands `\patterns` and `\hyphenation`—"pseudo" because these files are not read by TeX anymore, but by a tool of our own, called *inittrie*, written in C. These files, for which we recommend the file extension `.pat`, are compiled into compressed binary form (file extension `.hyp`) by *inittrie*. This binary form contains a certain number of tries, exactly as formerly stored in TeX format files.

Here is a detailed description of the ingredients of `.pat` files.

(left | right)hyphenmin
The primitives *lefthyphenmin* and *righthyphenmin* are used in TeX to specify the minimal number of letters to leave on a line, or to allow on the next line. In our case, these values are included in the patterns file, so that there is no need to worry about them in the TeX file, or Babel language file, etc.

equivalents
The argument of this command consists of a big number of character pairs, separated by blanks. These character pairs play the same role as *lccode* instructions in TeX. In each pair, the first character is a character considered to be a letter by the hyphenation algorithm, and the second one is its equivalence class. These characters are, of course, all written in Unicode UTF-8. In fig. 3 the reader can see this command displayed under Mac OS X.

patterns
Patterns are described in the same way as in TeX hyphenation files, with an extra convention: numbers between brackets specify the number of the hyphenation penalty register requested. So, for example, the hypothetical pattern

```
.con8s
```

for French language can be replaced by

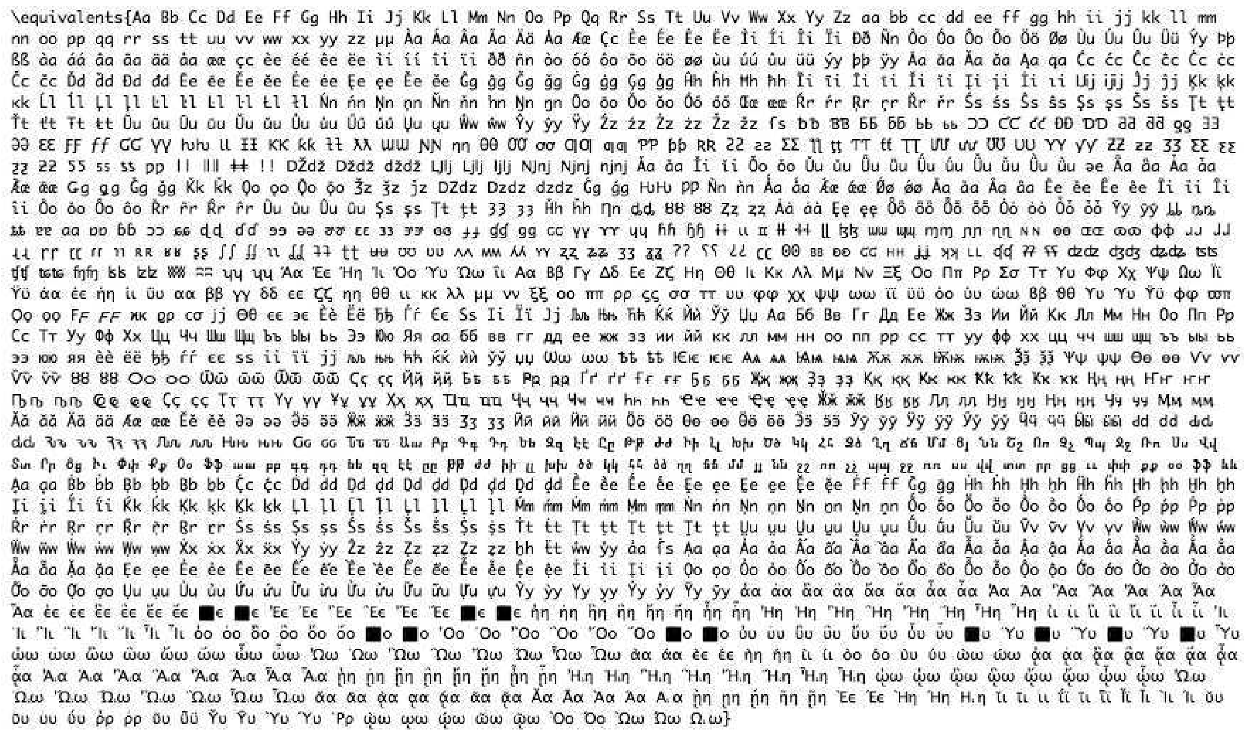
```
.con8[17]s
```

so that the value of hyphenation penalty register 17 will be used.

segmenthyphenpenalty
Using this pseudo-command one can specify the hyphenation penalty register to be used between word components. Default value is 1.

segmenthyphencharacter
Through this pseudo-command one can specify the hyphenation character to be used between word components. Default value is the hyphen. In cases like Thai, where we really segment a sentence into words (and words into syllables), the segment hyphenation character will be empty.

segmentpatterns
The syntax of the argument of this pseudo-command is the same as for *patterns*. These patterns will be used to obtain word components rather than syllables. The other two arguments specify the number of hyphen-



```
\equivalents{Aa Bb Cc Dd Ee Ff Gg Hh Ii Jj Kk Ll Mm Nn Oo Pp Qq Rr Ss Tt Uu Vv Ww Xx Yy Zz aa bb cc dd ee ff gg hh ii jj kk ll mm
nn oo pp qq rr ss tt uu vv ww xx yy zz µμ Àa Áa Âa Ãa Äa Åa Ææ Çc Èe Ée Êe Ëe Ìi Íi Îi Ïi Ðð Ñn Òo Óo Ôo Õo Öö Øø Ùu Úu Ûu Üü Ýy Þþ
ßß àa áá âa ãa ää åa ææ çc èe éé êe ëe ìi íí îi ïi ðð ñn òo óó ôo õo öö øø ùu úú ûu üü ýy þþ ÿy Āa āa Ăa ăa Ąa ąa Ćc ćc Ĉc ĉc Ċc ċc
Čc čc Ďd ďd Đd đd Ēe ēe Ĕe ĕe Ėe ėe Ęe ęe Ěe ěe Ĝg ĝg Ğg ğg Ġg ġg Ģg ģg Ĥh ĥh Ħh ħh Ĩi ĩi Īi īi Ĭi ĭi Įi įi İi ıi Ĳj ĳj Ĵj ĵj Ķk ķk
ĸk Ĺl ĺl Ļl ļl Ľl ľl Ŀl ŀl Łl łl Ńn ńn Ņn ņn Ňn ňn ŉn Ŋn ŋn Ōo ōo Ŏo ŏo Őő őő Œœ œœ Ŕr ŕr Ŗr ŗr Řr řr Śs śs Ŝs ŝs Şs şs Šs šs Ţt ţt
Ťt ťt Ŧt ŧt Ũu ũu Ūu ūu Ŭu ŭu Ůu ůu Űű űű Ųu ųu Ŵw ŵw Ŷy ŷy Ÿy Źz źz Żz żz Žz žz ſs ƀƀ ƁƁ ƂƂ ƃƃ ƄƄ ƅƅ ƆƆ ƇƇ ƈƈ ƉƉ ƊƊ ƋƋ ƌƌ ƍƍ ƎƎ
ƏƏ ƐƐ ƑƑ ƒƒ ƓƓ ƔƔ ƕƕ ƖƖ ƗƗ ƘƘ ƙƙ ƚƚ ƛƛ ƜƜ ƝƝ ƞƞ ƟƟ ƠƠ ơơ ƢƢ ƣƣ ƤƤ ƥƥ ƦƦ ƧƧ ƨƨ ƩƩ ƪƪ ƫƫ ƬƬ ƭƭ ƮƮ ƯƯ ưư ƱƱ ƲƲ ƳƳ ƴƴ ƵƵ ƶƶ ƷƷ ƸƸ ƹƹ
ƺƺ ƻƻ ƼƼ ƽƽ ƾƾ ƿƿ ǀǀ ǁǁ ǂǂ ǃǃ Ǆǆ ǅǆ ǆǆ Ǉǉ ǈǉ ǉǉ Ǌǌ ǋǌ ǌǌ Ǎa ǎa Ǐi ǐi Ǒo ǒo Ǔu ǔu Ǖu ǖu Ǘu ǘu Ǚu ǚu Ǜu ǜu ǝe Ǟa ǟa Ǡa ǡa
Ǣæ ǣæ Ǥg ǥg Ǧg ǧg Ǩk ǩk Ǫo ǫo Ǭo ǭo Ǯz ǯz ǰj Ǳdz ǲdz ǳdz Ǵg ǵg Ƕƕ Ƿƿ Ǹn ǹn Ǻa ǻa Ǽæ ǽæ Ǿø ǿø Ȁa ȁa Ȃa ȃa Ȅe ȅe Ȇe ȇe Ȉi ȉi Ȋi
ȋi Ȍo ȍo Ȏo ȏo Ȑr ȑr Ȓr ȓr Ȕu ȕu Ȗu ȗu Șs șs Țt țt Ȝȝ ȝȝ Ȟh ȟh Ƞn ȡȡ ȢȢ ȣȣ Ȥz ȥz Ȧa ȧa Ȩę ȩę Ȫö ȫö Ȭõ ȭõ Ȯo ȯo Ȱȱ ȱȱ Ȳy ȳy ȴȴ ȵȵ
ȶȶ ɐɐ ɑɑ ɒɒ ɓɓ ɔɔ ɕɕ ɖɖ ɗɗ ɘɘ əə ɚɚ ɛɛ ɜɜ ɝɝ ɞɞ ɟɟ ɠɠ ɡɡ ɢɢ ɣɣ ɤɤ ɥɥ ɦɦ ɧɧ ɨɨ ɩɩ ɪɪ ɫɫ ɬɬ ɭɭ ɮɮ ɯɯ ɰɰ ɱɱ ɲɲ ɳɳ ɴɴ ɵɵ ɶɶ ɷɷ ɸɸ ɹɹ ɺɺ
ɻɻ ɼɼ ɽɽ ɾɾ ɿɿ ʀʀ ʁʁ ʂʂ ʃʃ ʄʄ ʅʅ ʆʆ ʇʇ ʈʈ ʉʉ ʊʊ ʋʋ ʌʌ ʍʍ ʎʎ ʏʏ ʐʐ ʑʑ ʒʒ ʓʓ ʔʔ ʕʕ ʖʖ ʗʗ ʘʘ ʙʙ ʚʚ ʛʛ ʜʜ ʝʝ ʞʞ ʟʟ ʠʠ ʡʡ ʢʢ ʣʣ ʤʤ ʥʥ ʦʦ
ʧʧ ʨʨ ʩʩ ʪʪ ʫʫ ʬʬ ʭʭ ʮʮ ʯʯ Άα Έε Ήη Ίι Όο Ύυ Ώω ΐι Αα Ββ Γγ Δδ Εε Ζζ Ηη Θθ Ιι Κκ Λλ Μμ Νν Ξξ Οο Ππ Ρρ Σσ Ττ Υυ Φφ Χχ Ψψ Ωω Ϊι
Ϋυ άα έε ήη ίι ΰυ αα ββ γγ δδ εε ζζ ηη θθ ιι κκ λλ μμ νν ξξ οο ππ ρρ ςς σσ ττ υυ φφ χχ ψψ ωω ϊι ϋυ όο ύυ ώω ϐβ ϑθ ϒυ ϓυ ϔυ ϕφ ϖπ
Ϙϙ ϙϙ Ϝϝ ϝϝ ϰκ ϱρ ϲσ ϳj ϴθ ϵε ϶϶ Ѐè Ёë Ђђ Ѓґ Єє Ѕs Іi Її Јj Љљ Њњ Ћћ Ќќ Ѝй Ўў Џџ Аа Бб Вв Гг Дд Ее Жж Зз Ии Йй Кк Лл Мм Нн Оо Пп Рр
Сс Тт Уу Фф Хх Цц Чч Шш Щщ Ъъ Ыы Ьь Ээ Юю Яя аа бб вв гг дд ее жж зз ии йй кк лл мм нн оо пп рр сс тт уу фф хх цц чч шш щщ ъъ ыы ьь
ээ юю яя èè ёë ђђ ѓґ єє ѕs іi її јj љљ њњ ћћ ќќ ѝй ўў џџ Ѡѡ ѡѡ Ѣѣ ѣѣ Ѥѥ ѥѥ Ѧѧ ѧѧ Ѩѩ ѩѩ Ѫѫ ѫѫ Ѭѭ ѭѭ Ѯѯ ѯѯ Ѱѱ ѱѱ Ѳѳ ѳѳ Ѵѵ ѵѵ
Ѷѵ ѷѵ Ѹѹ ѹѹ Ѻѻ ѻѻ Ѽѽ ѽѽ Ѿѿ ѿѿ Ҁҁ ҁҁ Ҋҋ ҋҋ Ҍҍ ҍҍ Ҏҏ ҏҏ Ґґ ґґ Ғғ ғғ Ҕҕ ҕҕ Җҗ җҗ Ҙҙ ҙҙ Ққ ққ Ҝҝ ҝҝ Ҟҟ ҟҟ Ҡҡ ҡҡ Ңң ңң Ҥҥ ҥҥ
Ҧҧ ҧҧ Ҩҩ ҩҩ Ҫҫ ҫҫ Ҭҭ ҭҭ Үү үү Ұұ ұұ Ҳҳ ҳҳ Ҵҵ ҵҵ Ҷҷ ҷҷ Ҹҹ ҹҹ Һһ һһ Ҽҽ ҽҽ Ҿҿ ҿҿ Ӂӂ ӂӂ Ӄӄ ӄӄ Ӈӈ ӈӈ Ӌӌ ӌӌ Ӎӎ ӎӎ
Ӑӑ ӑӑ Ӓӓ ӓӓ Ӕӕ ӕӕ Ӗӗ ӗӗ Әә әә Ӛӛ ӛӛ Ӝӝ ӝӝ Ӟӟ ӟӟ Ӡӡ ӡӡ Ӣӣ ӣӣ Ӥӥ ӥӥ Ӧӧ ӧӧ Өө өө Ӫӫ ӫӫ Ӭӭ ӭӭ Ӯӯ ӯӯ Ӱӱ ӱӱ Ӳӳ ӳӳ Ӵӵ ӵӵ Ӹӹ ӹӹ ԁԁ ԁԁ ԃԃ
ԃԃ Ԅԅ ԅԅ Ԇԇ ԇԇ Ԉԉ ԉԉ Ԋԋ ԋԋ Ԍԍ ԍԍ Ԏԏ ԏԏ Աա Բբ Գգ Դդ Եե Զզ Էէ Ըը Թթ Ժժ Իի Լլ Խխ Ծծ Կկ Հհ Ձձ Ղղ Ճճ Մմ Յյ Նն Շշ Ոո Չչ Պպ Ջջ Ռռ Սս Վվ
Տտ Րր Ցց Ււ Փփ Քք Օօ Ֆֆ աա բբ գգ դդ եե զզ էէ ըը թթ ժժ իի լլ խխ ծծ կկ հհ ձձ ղղ ճճ մմ յյ նն շշ ոո չչ պպ ջջ ռռ սս վվ տտ րր ցց ււ փփ քք օօ ֆֆ կk
Ḁa ḁa Ḃb ḃb Ḅb ḅb Ḇb ḇb Ḉc ḉc Ḋd ḋd Ḍd ḍd Ḏd ḏd Ḑd ḑd Ḓd ḓd Ḕe ḕe Ḗe ḗe Ḙe ḙe Ḛe ḛe Ḝe ḝe Ḟf ḟf Ḡg ḡg Ḣh ḣh Ḥh ḥh Ḧh ḧh Ḩh ḩh Ḫh ḫh
Ḭi ḭi Ḯi ḯi Ḱk ḱk Ḳk ḳk Ḵk ḵk Ḷl ḷl Ḹl ḹl Ḻl ḻl Ḽl ḽl Ḿm ḿm Ṁm ṁm Ṃm ṃm Ṅn ṅn Ṇn ṇn Ṉn ṉn Ṋn ṋn Ṍo ṍo Ṏo ṏo Ṑo ṑo Ṓo ṓo Ṕp ṕp Ṗp ṗp
Ṙr ṙr Ṛr ṛr Ṝr ṝr Ṟr ṟr Ṡs ṡs Ṣs ṣs Ṥs ṥs Ṧs ṧs Ṩs ṩs Ṫt ṫt Ṭt ṭt Ṯt ṯt Ṱt ṱt Ṳu ṳu Ṵu ṵu Ṷu ṷu Ṹu ṹu Ṻu ṻu Ṽv ṽv Ṿv ṿv Ẁw ẁw Ẃw ẃw
Ẅw ẅw Ẇw ẇw Ẉw ẉw Ẋx ẋx Ẍx ẍx Ẏy ẏy Ẑz ẑz Ẓz ẓz Ẕz ẕz ẖh ẗt ẘw ẙy ẚa ẛs Ạa ạa Ảa ảa Ấa ấa Ầa ầa Ẩa ẩa Ẫa ẫa Ậa ậa Ắa ắa Ằa ằa Ẳa ẳa
Ẵa ẵa Ặa ặa Ẹe ẹe Ẻe ẻe Ẽe ẽe Ếe ếe Ềe ềe Ểe ểe Ễe ễe Ệe ệe Ỉi ỉi Ịi ịi Ọo ọo Ỏo ỏo Ốo ốo Ồo ồo Ổo ổo Ỗo ỗo Ộo ộo Ớo ớo Ờo ờo Ởo ởo
Ỡo ỡo Ợo ợo Ụu ụu Ủu ủu Ứu ứu Ừu ừu Ửu ửu Ữu ữu Ựu ựu Ỳy ỳy Ỵy ỵy Ỷy ỷy Ỹy ỹy ἀα ἁα ἂα ἃα ἄα ἅα ἆα ἇα Ἀα Ἁα Ἂα Ἃα Ἄα Ἅα Ἆα
Ἇα ἐε ἑε ἒε ἓε ἔε ἕε ■ε ■ε Ἐε Ἑε Ἒε Ἓε Ἔε Ἕε ■ε ■ε ἠη ἡη ἢη ἣη ἤη ἥη ἦη ἧη Ἠη Ἡη Ἢη Ἣη Ἤη Ἥη Ἦη Ἧη ἰι ἱι ἲι ἳι ἴι ἵι ἶι ἷι Ἰι
Ἱι Ἲι Ἳι Ἴι Ἵι Ἶι Ἷι ὀο ὁο ὂο ὃο ὄο ὅο ■ο ■ο Ὀο Ὁο Ὂο Ὃο Ὄο Ὅο ■ο ■ο ὐυ ὑυ ὒυ ὓυ ὔυ ὕυ ὖυ ὗυ ■υ Ὑυ ■υ Ὓυ ■υ Ὕυ ■υ Ὗυ
ὠω ὡω ὢω ὣω ὤω ὥω ὦω ὧω Ὠω Ὡω Ὢω Ὣω Ὤω Ὥω Ὦω Ὧω ὰα άα ὲε έε ὴη ήη ὶι ίι ὸο όο ὺυ ύυ ὼω ώω ᾀα ᾁα ᾂα ᾃα ᾄα ᾅα ᾆα
ᾇα ᾈα ᾉα ᾊα ᾋα ᾌα ᾍα ᾎα ᾏα ᾐη ᾑη ᾒη ᾓη ᾔη ᾕη ᾖη ᾗη ᾘη ᾙη ᾚη ᾛη ᾜη ᾝη ᾞη ᾟη ᾠω ᾡω ᾢω ᾣω ᾤω ᾥω ᾦω ᾧω ᾨω
ᾩω ᾪω ᾫω ᾬω ᾭω ᾮω ᾯω ᾰα ᾱα ᾲα ᾳα ᾴα ᾶα ᾷα Ᾰα Ᾱα Ὰα Άα ᾼα ῂη ῃη ῄη ῆη ῇη Ὲε Έε Ὴη Ήη ῌη ῐι ῑι ῒι ΐι ῖι ῗι Ῐι Ῑι Ὶι Ίι ῠυ
ῡυ ῢυ ΰυ ῤρ ῥρ ῦυ ῧυ Ῠυ Ῡυ Ὺυ Ύυ Ῥρ ῲω ῳω ῴω ῶω ῷω Ὸο Όο Ὼω Ώω ῼω}
```

Figure 3: Table of equivalents

ation penalty register to be used, and the requested secondary hyphenation character.

transitions

To use a finite-state engine, we need the commands `\transitions`, `\references` and `\segments` instead of the command `\segmentpatterns`. In the argument of the `\transitions` command we will write transitions in the syntax given above: strings `BEGIN` and `END` for beginnings and ends of words, numbers for all other transitions, the string `->` between origin and destination of a transition. Example: `BEGIN->1 BEGIN->2 1->1 1->2 2->*1 2->*2 2->3 2->END 3->*1 3->*2 3->3 3->END` for the *SiSiSi* model. The asterisk means that the given transition produces a new segment. A destination followed by an `F` means that there is a filter: the transition occurs only if origin and destination belong to the same family.

references

This command contains "references". A reference is a set of families. The idea is the following: a segment very often belongs to several families (meaning that it can be combined with many other segments, in order to form components and words). Instead of writing all the families to which each segment belongs, we will in fact use a single number. This number will be an index to the set of families to which it belongs, its reference.

The syntax is shown by the following example:

```
2368=/843/844/845/921/943
```

meaning that segments followed by number 2368 belong to families 843, 844, 845, 921 and 943. When checking whether a transition is allowed, our algorithm will not check if the references are the same, but rather if they have a non-empty common set of families. References are separated by blanks.

segments

Segments are classified by the state to which they belong. The argument of `\segments` looks like:

```
\segments{
1: %prefixes
a2083 abba2084 agyon2084 alá2084 b2085
be2084 bele2084 benn2084 benn-2084
billió2086 c2087 d2088 e2089 egy2086
egybe2084 el2084 ...
2: %original stems
aba3 abafala3 abafalva3 abaffy5 abafi6
abafája3 abajgat8 abakteriális9 ...
3: %pre-altered stems
...
4: %post-altered stems
...
ab2 abafal2 abafalv4 abafáj2 abajga7
abalehot2 abar2 abaújharaszt14
abaújszakal18 abaújszin2 abaújtorn2
abd2 abell25 abelov2 ...
5: %prepost-altered stems
...
6: %stems without affixes
...
7: %suffixes
abbak2137 abbakat2138 abbakba2138
abbakban2138 abbakból2138 abbakhoz2138
abbakig2138 abbakkal2138 abbakká2138
abbaknak2138 ...
}
```

A number followed by a colon denotes a state. Segments are separated by blanks. They are followed by reference numbers.

If the finite-state engine does not require family filtering, then the *references* command will be empty and segments will not be followed by any number.

lastsegmentpriority
Whenever this option is included in the patterns file, the hyphenation points in the last segment have their hyphenation penalty registers increased by a given amount.

excentricity
Gives the excentricity factor.

References

[1] Antoš D., "PATLIB, Pattern Manipulation Library", Master Thesis, Masaryk University Brno, 2001. `http://www.fi.muni.cz/~xantos/patlib/thesis/thesis-p.pdf`

[2] Barth W., Steiner H., "Deutsche Silbentrennung für TEX 3.1", *Die TEXnische Komödie*, 1, 1992, pp. 33-35. `http://www.dante.de/dante/DTK/dtk92_1/dtk92_1_barth_steiner_deutsche.html` and `http://www.ads.tuwien.ac.at/research/SiSiSi.html`

[3] Haralambous Y., Bella G., "Injecting Information into Atomic Units of Text", in Proceedings of the ACM Symposium on Document Engineering, Bristol, 2005. `http://omega.enstb.org/yannis/pdf/fp10174-haralambous.pdf`

[4] Haralambous Y., "Voyage au centre de TEX : composition, paragraphage, césure", *Cahiers GUTenberg* 44-45, 2004, pp. 3-53. `http://omega.enstb.org/yannis/pdf/voyage.pdf`

[5] Haralambous Y., "From Unicode to Typography, a Case Study: the Greek Script", Proceedings of International Unicode Conference XIV, Boston, 1999, pp. b.10.1–b.10.36. `http://omega.enstb.org/yannis/pdf/boston99.pdf`

[6] Haralambous Y., "A Small Tutorial on the Multilingual Features of PATGEN2", in electronic form, available from CTAN as `info/patgen2.tutorial`, 1994.

[7] Haralambous Y., "Using PATGEN to Create Welsh Patterns", submitted to *TUGboat*, 1993.

[8] Haralambous Y., "Hyphenation Patterns for Ancient Greek and Latin", *TUGboat* 13 (4), 1992, pp. 457-469. `http://omega.enstb.org/yannis/pdf/ancgreek92.pdf`

[9] Knuth D.E., *Computers & Typesetting, Vol. B: TEX, The Program*, Addison-Wesley, 1986.

[10] Raichle B., "Hyphenation patterns for words containing explicit hyphens", `CTAN/language/hyphenation/hypht1.tex`, 1997.

[11] Scannell K, "Hyphenation Patterns for Minority Languages", *TUGboat* 24 (2), 2003, pp. 236–239. `http://www.tug.org/TUGboat/Articles/tb24-2/tb77scannell.pdf`

[12] Sojka P., "Hyphenation on Demand", *TUGboat* 20 (3), 1999. `http://www.tug.org/TUGboat/Articles/tb20-1/tb62sched.pdf`

[13] Sojka P., Ševeček P., "Hyphenation in TEX — Quo Vadis?" *TUGboat* 16 (3), pp. 280–289, 1995. `http://www.tug.org/TUGboat/Articles/tb16-3/tb48soj1.pdf`

[14] Sojka P., "Notes on Compound Word Hyphenation in TEX", *TUGboat* 16 (3), pp. 290–297, 1995. `http://www.tug.org/TUGboat/Articles/tb16-3/tb48soj2.pdf`

Abstracts

Typesetting the Qur'an and its specific challenges to the TEX family

Hossam A.H. Fahmy, Cairo University

AlQalam ("the pen" in Arabic) is our freely available system intended for typesetting the Qur'an, other traditional texts, and any publications in the languages using the Arabic script. From a typographical point of view, the Qur'an is one of the most demanding texts. However, there is a long historical record of excellent quality materials (manuscripts and recent printings) to guide the work on a system to typeset it. Such a system, once complete, can easily typeset any work using the Arabic script, including those with mixed languages.

(The full paper will be printed in the proceedings of the TUG 2006 conference. *Ed.*)

How to deal with TEX in unfriendly situations

Hans Hagen, Pragma ADE

Tools and methods for dealing with TEX in unfriendly situations, such as multiple trees, potentially conflicting environments, and strictly regulated web services.

Making better PDF

Hartmut Henkel, pdfTEX team

Under normal circumstances, once the TEX part of pdfTEX is happy with the input, and all the required stuff like fonts and graphics is available, pdfTEX produces a valid PDF file. This talk is about PDF output, looking under the hood.

To learn about the general PDF file format and structure, we have a look at a PDF file generated from a short LATEX example. We follow the steps of a simple virtual PDF viewer from opening the file over collecting required resources until text from a typical page is placed on the output medium. We see how the PDF file is made up from many objects that can be found in the file by consulting a cross reference table. The most important building blocks for objects (dictionaries and streams) and data structures are described, and the basic operators for coordinate transforms, font selection and glyph placement are discussed.

Then we have a look at how pdfTEX implements precise glyph placement, taking into account that most movements are incremental, that there are two different coordinate systems involved (one for TEX's internal calculations and one for the PDF output representation), and that coordinate values in PDF files are rounded to only a few decimal digits. As a PDF viewer knows only these rounded numbers (and not the exact TEX ones), there could be the risk of error accumulation; pdfTEX prevents this by keeping track of the rounded numbers it has output to the PDF file.

In the last part of the talk we do a little detective work: if one inspects a page stream generated by pdfTEX one can often spot tiny correction terms '`)1(`' and '`)-1(`' in the glyph placement array that, interestingly, are there only for the Computer Modern fonts, not for standard PostScript fonts like Times-Roman. The origin of these corrections is traced to the fact that the CM fonts are not designed on a 1/1000 fontsize raster, as they predate PostScript. Finally, a tiny patch to pdfTEX is presented that makes these correction terms happen much less frequently while keeping precision, which leads to slightly tidier and smaller PDF files.

BibTEX, MlBibTEX, and bibliography styles

Jean-Michel Hufflen, University of Franche-Comté

The first part of this talk about BibTEX will focus on some difficult points related to the syntax of bibliography files, e.g., the specification of person and organisation names. In addition, we show how some successors of BibTEX — the BibTEX8, Bibulus, and MlBibTEX programs — improve them. In a second part, we explain how bibliography styles are built. Some demonstrations of the BibTEX program are given, and some technical points could be made clearer by using some functions belonging to MlBibTEX.

(The full paper was printed in the proceedings of the BachoTEX 2006 conference. *Ed.*)

Typography — the art of the letter system

István Radó, Radó Kiadó and Szolgáltató MC

This talk will discuss the dialectic of the typographic print and the service of the ergonomics of reading, as well as the damage caused by malformed prints resulting from its partial or complete negligence — in the context of European Roman letters.

Writing is conserving human thought in a systematic order. Printing is the generator of intellect, and occasionally a bearer of real knowledge, too. For over a thousand years letters have been the means of representation of the creative human intellect on papyrus, then paper, and for some 30 years now on the computer screen, too.

The term 'letter' signifies the so-called Roman letters that emerged from ancient Greek and Roman culture. It was this type of letter that had as its destiny to be a primary carrier of knowledge and culture in Europe, and thus it has served a worldwide technical civilisation emerging from European culture, with all its well-known blessings and curses.

Today 60% of humanity is illiterate, and half of the rest is functionally illiterate. Halving again, we find the percentage of those who do not read anything except the news and tabloids — therefore it is a mere 20% of the entire population of the earth that seeks knowledge from letters. And there is an evergrowing proportion within this 20% representing Chinese speakers! This latter fact provides a peculiar (and alarming) future to the Roman letter.

However, the documentation of the past, present and near future remains a task for European Roman letters and numbers nearly exclusively. The appearance and the point of this form of expression is the dialectic harmony of form and content, or an apparent and highly disturbing lack of it. It is easy to express the gist of a given idea in letters: unity of thought developing from contrapositions, opinion, reasoning, assertion, informative teaching, the obscuring of the point, plagiarism, incitement, forgery, lying etc.

Good, well-made and intelligent typography gives emphasis and order to the thought it represents. A printing type that is ostentatious, or one that neglects or consciously infringes a moderate proportionality is always a sign of the superficiality and false nature of the message itself. The blissful development of PCs regards as uniform and (perhaps unduly) degrades, too, all prints produced by repetitious use of the same font types and groups. The chaotic and disproportionate location of font size and spacing of lines are evidence of the damage made to the message, or its very worthlessness.

pdfTEX — what was, is and will be

Martin Schröder, pdfTEX team

This talk is a review of key pdfTEX features and primitives presented on a timeline from the beginning, including present versions (1.30 and 1.40), and through the features planned for the near future.

LATEX programming tutorial

Péter Szabó, Budapest University of Technology and Economics

This tutorial is a practical introduction to LATEX programming: implementing new features (writing LATEX packages), writing packages accepting options, changing existing features, finding out what commands to change, finding the file containing the definition of the command, overriding the definition, extending the definition, debugging, writing code independent of catcode changes, string processing and `.aux` file tricks.

Calendar

2006

Oct 12 – 14	Guild of Book Workers 100th Anniversary Exhibition: A traveling juried exhibition of books by members of the Guild of Book Workers. Grolier Club, New York City. Sites and dates are listed at `http://palimpsest.stanford.edu/byorg/gbw`.
Oct 16 – 18	Fifth Annual St. Bride Conference, "Fast Type, Slow Type", Birmingham, England. Celebrating the 300th birthday of John Baskerville. For information, visit `http://www.creativepro.com/story/news/23853.html`.
Oct 20	UK TUG Joint workshop with the London Mathematical Society, "Living and Working with LaTeX", and Annual general meeting, London. For information, visit `http://uk.tug.org/`.
Oct 20 – 22	The Fourth International Conference on the Book, "Save, Change or Discard: Tradition and Innovation in the World of Books", Emerson College, Boston, Massachusetts. For information, visit `http://book-conference.com`.
Oct 21	GuIT meeting 2006 (Gruppo utilizzatori Italiani di TeX), Pisa, Italy. For information, visit `http://www.guit.sssup.it/GuITmeeting/2006/2006.en.php`.

TUG 2006
Digital Typography & Electronic Publishing: Localization & Internationalization, Marrakesh, Morocco.

Nov 7 – 8	Pre-conference tutorials.
Nov 9 – 11	The 27th annual meeting of the TeX Users Group. For information, visit `http://www.tug.org/tug2006`.

Nov 18	NTG 38th meeting, Zeist, Netherlands. For information, visit `http://www.ntg.nl/bijeen/bijeen38.html`.
Dec 4 – 7	<XML2006>, Boston, Massachusetts. For information, visit `http://2006.xmlconference.org/`.

2007

Feb 1	**TUG Election:** Deadline for nominations (see p.107).
Mar 7 – 9	DANTE 2007, 36th meeting, Westfälische Wilhelms-Universitẗ, Münster, Germany. For information, visit `http://www.dante.de/dante2007`.
Mar 24 – 25	First international CONTEXT User Meeting, Epen, The Netherlands. For information, visit `http://context.aanhet.net/epen2007`.
Apr 28 – May 2	17th EuroTeX Conference + 15th BachoTeX Conference = EuroBachoTeX 2007, Bachotek, Poland. For information, visit `http://www.gust.org.pl/BachoTeX/EuroBachoTeX2007`.

TUG 2007
Practicing TeX,
San Diego, California.

Jul 17	Workshops (free for attendees).
Jul 18 – 20	The 28th annual meeting of the TeX Users Group. For information, visit `http://www.tug.org/tug2007`.

Aug 5 – 9	SIGGRAPH 2007, San Diego, California. For information, visit `http://www.siggraph.org/s2007/`.
Aug 6 – 10	*Extreme* Markup Languages 2007, Montréal, Québec. For information, visit `http://www.extrememarkup.com/extreme/`.

Status as of 15 October 2006

For additional information on TUG-sponsored events listed here, contact the TUG office (+1 503 223-9994, fax: +1 206 203-3960, e-mail: `office@tug.org`). For events sponsored by other organizations, please use the contact address provided.

An updated version of this calendar is online at `http://www.tug.org/calendar/`.

Additional type-related events are listed in the Typophile calendar, at `http://www.icalx.com/html/typophile/month.php?cal=Typophile`.

2007 TEX Users Group election

Steve Peter, for the Elections Committee

The positions of TUG President and several members of the Board of Directors will be open as of the 2007 Annual Meeting.

The directors whose terms will expire in 2007: Barbara Beeton, Jon Breitenbucher, Kaja Christiansen, Susan DeMeritt, Ross Moore, Cheryl Ponchin, Samuel Rhoads, and Philip Taylor. One additional director position is currently unoccupied.

Continuing directors, with terms ending in 2009, are: Steve Grathwohl, Jim Hefferon, Klaus Höppner, Arthur Ogawa, Steve Peter, and David Walden.

The election to choose the new President and Board members will be held in spring of 2007. Nominations for these openings are now invited.

The Bylaws provide that "Any member may be nominated for election to the office of TUG President/to the Board by submitting a nomination petition in accordance with the TUG Election Procedures. Election ... shall be by written mail ballot of the entire membership, carried out in accordance with those same Procedures." The term of President is two years, of Director four years.

The name of any member may be placed in nomination for election to one of the open offices by submission of a petition, signed by two other members in good standing, to the TUG office.

Nomination forms, along with all the details, are available via the TUG Web pages at `http://tug.org/election`, or from the TUG office.

Along with a nomination form, each candidate must supply a passport-size photograph, a short biography, and a statement of intent to be included with the ballot; the biography and statement of intent together may not exceed 400 words. Also, a candidate's membership dues for 2007 are expected to be paid by the nomination deadline.

The deadline for receipt at the TUG office of nomination forms and ballot information:
1 February 2007.

Institutional Members

Aalborg University, Department of Mathematical Sciences, *Aalborg, Denmark*

American Mathematical Society, *Providence, Rhode Island*

Banca d'Italia, *Roma, Italy*

Center for Computing Science, *Bowie, Maryland*

Certicom Corp., *Mississauga, Ontario Canada*

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 Defense Analyses, Center for Communications Research, *Princeton, New Jersey*

MacKichan Software, *Washington/New Mexico, USA*

Marquette University, Department of Mathematics, Statistics and Computer Science, *Milwaukee, Wisconsin*

Masaryk University, Faculty of Informatics, *Brno, Czechoslovakia*

New York University, Academic Computing Facility, *New York, New York*

Princeton University, Department of Mathematics, *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*

United States Environmental Protection Agency, *Narragansett, Rhode Island*

University College, Cork, Computer Centre, *Cork, Ireland*

University of Delaware, Computing and Network Services, *Newark, Delaware*

Université Laval, *Ste-Foy, Québec, Canada*

University of Oslo, Institute of Informatics, *Blindern, Oslo, Norway*

Vanderbilt University, *Nashville, Tennessee*

TeX Consultants

The information here comes from the consultants themselves. We do not include information we know to be false, but we cannot check out any of the information; we are transmitting it to you as it was given to us and do not promise it is correct. Also, this is not an official endorsement of the people listed here. We provide this list to enable you to contact service providers and decide for yourself whether to hire one.

TUG also provides an online list of consultants at `http://tug.org/consultants.html`. If you'd like to be listed, please fill out the form at `https://www.tug.org/consultants/listing.html` or email us at `consult-admin@tug.org`. To place a larger ad in *TUGboat*, please see `http://tug.org/TUGboat/advertising.html`.

Kinch, Richard J.
7890 Pebble Beach Ct
Lake Worth, FL 33467
561-966-8400
Email: `kinch (at) truetex.com`
Publishes TrueTeX, a commercial implementation of TeX and LaTeX. Custom development for TeX-related software and fonts.

Martinez, Mercè Aicart
C/Tarragona 102 4^o 2^a
08015 Barcelona, Spain
+34 932267827
Email: `m.aicart (at) menta.net`
Web: `www.edilatex.com/index_eng.html`
We provide, at reasonable low cost, TeX and LaTeX typesetting services to authors or publishers world-wide. We have been in business since the beginning of 1990.

MCR Inc.
731 Beta Drive #G
Mayfield Village, OH 44143
(440) 484-3010; fax: (440) 484-3020
Email: `sales (at) mcr-inc.com`
Web: `www.mcr-inc.com`
Contract typesetting/printing services.

MicroPress Inc.
68-30 Harrow Street
Forest Hills, NY 11375
+1 718-575-1816; fax: +1 718-575-8038
Email: `support (at) micropress-inc.com`
Web: `www.micropress-inc.com`
Makers of VTeX, fully integrated TeX system running on Windows. VTeX system is capable of one-pass output of PDF, PS, SVG and HTML; VTeX IDE includes Visual Tools for writing equations, function plots and other enhancements and a large number of fonts, many not available elsewhere. Makers of many new and unique mathematical font families for use with TeX, see `http://www.micropress-inc.com/fonts`. Makers of microIMP, a fully WYSIWYG LaTeX-based Word Processor. microIMP supports TeX and AMSTeX math, lists, tables, slides, trees, graphics inclusion, many languages and much else — all without any need for knowing TeX commands, see `http://www.microimp.com`. See our web page for other products and services. Serving TeX users since 1989.

Peter, Steve
310 Hana Road
Edison, NJ 08817
+1 (732) 287-5392
Email: `speter (at) dandy.net`
Specializing in foreign language, linguistic, and technical typesetting using TeX, LaTeX, and ConTeXt, I have typeset books for Oxford University Press, Routledge, and Kluwer, and have helped numerous authors turn rough manuscripts, some with dozens of languages, into beautiful camera-ready copy. I have extensive experience in editing, proofreading, and writing documentation. I also tweak and design fonts. I have an MA in Linguistics from Harvard University and live in the New York metro area.

Veytsman, Boris
2239 Double Eagle Ct.
Reston, VA 20191
(703) 860-0013
Email: `borisv (at) lk.net`
Web: `http://users.lk.net/~borisv`
TeX/LaTeX consulting. Integration with databases, full automated document preparation systems, conversions and more.

Table of Contents (ordered by difficulty)

TUGboat

Volume 27, Number 1 / 2006
EuroTEX 2006 Conference Proceedings

Table of Contents (ordered by difficulty)